



(19) **United States**

(12) **Patent Application Publication**

MITCHELL et al.

(10) **Pub. No.: US 2002/0113816 A1**

(43) **Pub. Date: Aug. 22, 2002**

(54) **METHOD AND APPARATUS PROVIDING A GRAPHICAL USER INTERFACE FOR REPRESENTING AND NAVIGATING HIERARCHICAL NETWORKS**

(76) Inventors: **FREDERICK H. MITCHELL**,
NASHUA, NH (US); **DAVID K.**
BAINBRIDGE, ACTON, MA (US)

Correspondence Address:
HAMILTON, BROOK, SMITH & REYNOLDS,
P.C.
530 VIRGINIA ROAD
P.O. BOX 9133
CONCORD, MA 01742-9133 (US)

(*) Notice: This is a publication of a continued prosecution application (CPA) filed under 37 CFR 1.53(d).

(21) Appl. No.: **09/208,554**

(22) Filed: **Dec. 9, 1998**

Publication Classification

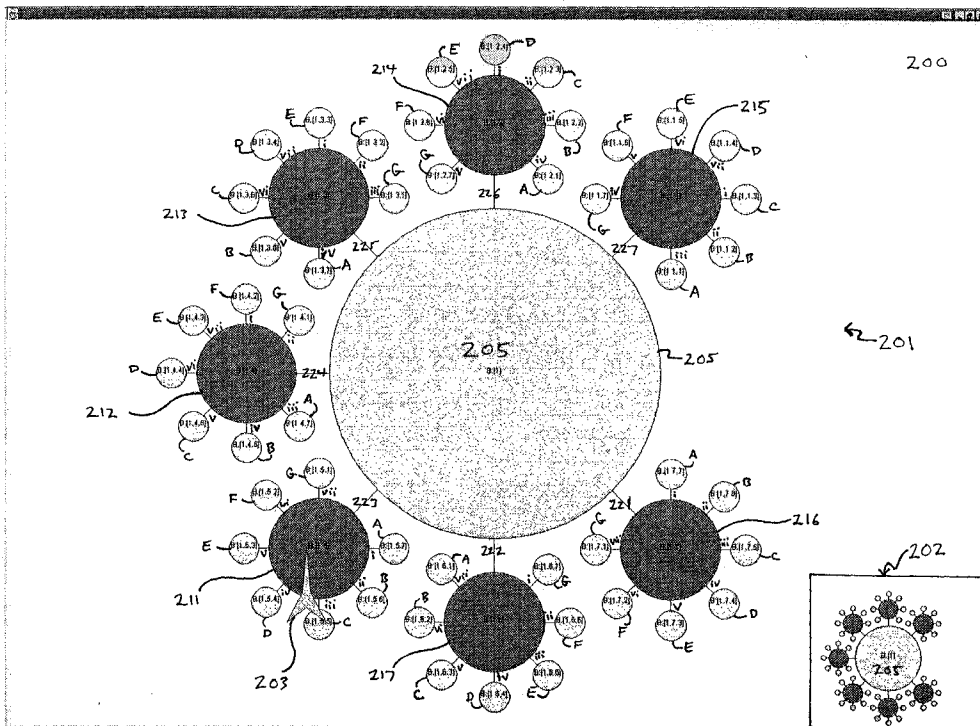
(51) Int. Cl.⁷ **G09G 5/00**

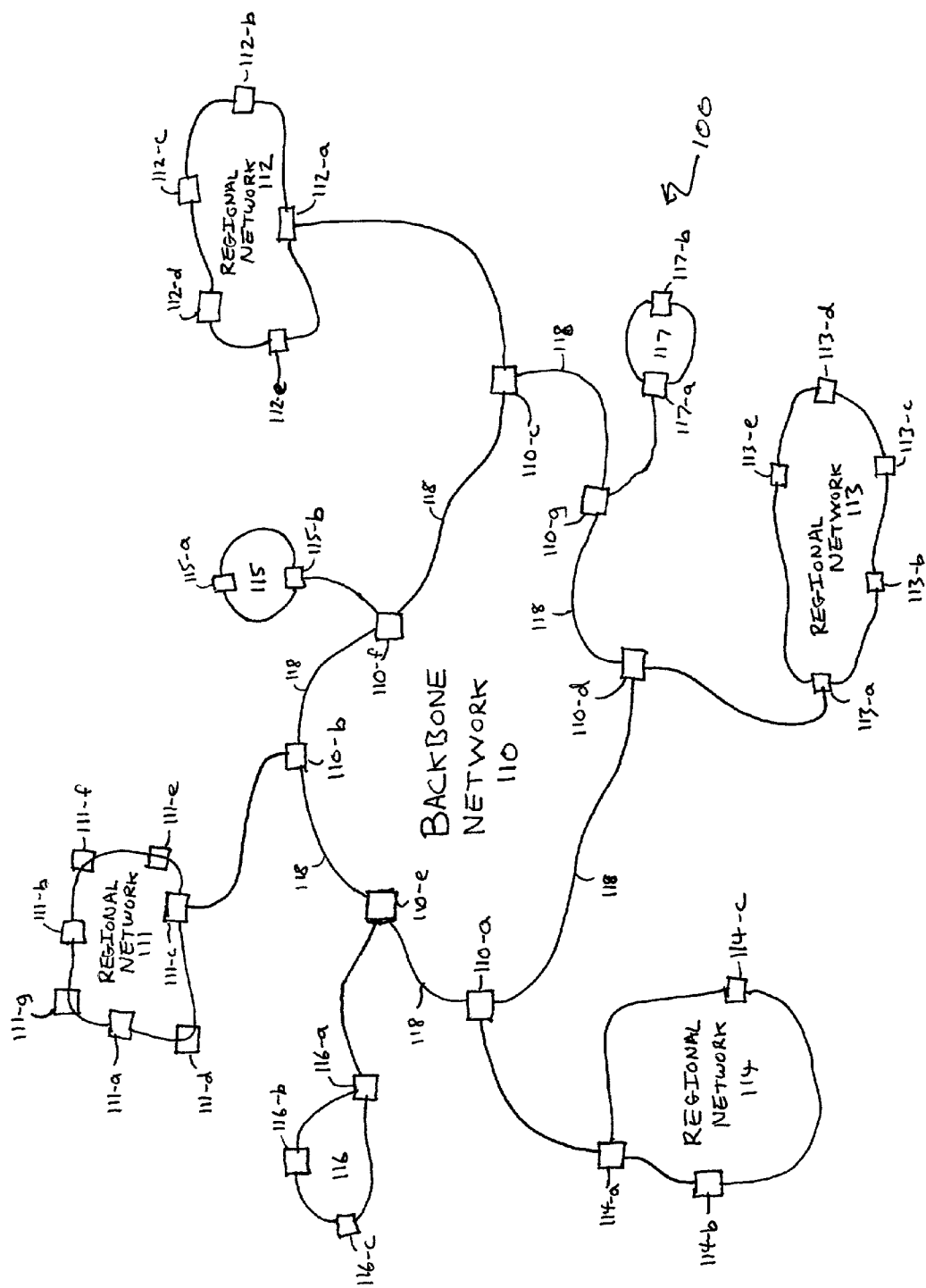
(52) U.S. Cl. **345/734**

(57) **ABSTRACT**

A method and apparatus are provided which present hierarchical data to a user via a graphical user interface. A

preferred embodiment represents hierarchical data related to a computer network and is provided to a user as part of a network management software application. In the interface, hierarchical data is represented by nodes, beginning with one or more top nodes and extending into lower hierarchical levels by the display of child nodes, child's child nodes, and so forth. The arrangement of nodes on the graphical user interface is such that scaling portrays the various hierarchical levels, and nodes do not spatially interfere with one another. Navigation through the hierarchical data is provided by allowing the user to select any visible node, at which point a zoom-in or zoom-out view to the selected node as a centrally located node on the interface is performed. Child nodes at lower hierarchical levels that were not visible before selection are then made visible up to a predetermined number of levels within the hierarchy. A map is provided on the interface which allows a user to graphically comprehend the present location of all nodes displayed on the interface in relation to their position within the overall hierarchy. As applied to network management, the interface allows errors in low level devices within a network to be visually propagated up to the upper levels of the hierarchy, for display to a user viewing only the top levels. The interface and computing system configured according to the invention overcomes problems of prior art systems which obscure a user location within a large hierarchy and the invention overcomes significant navigation problems of prior art systems.





PRIOR ART
FIG. 1

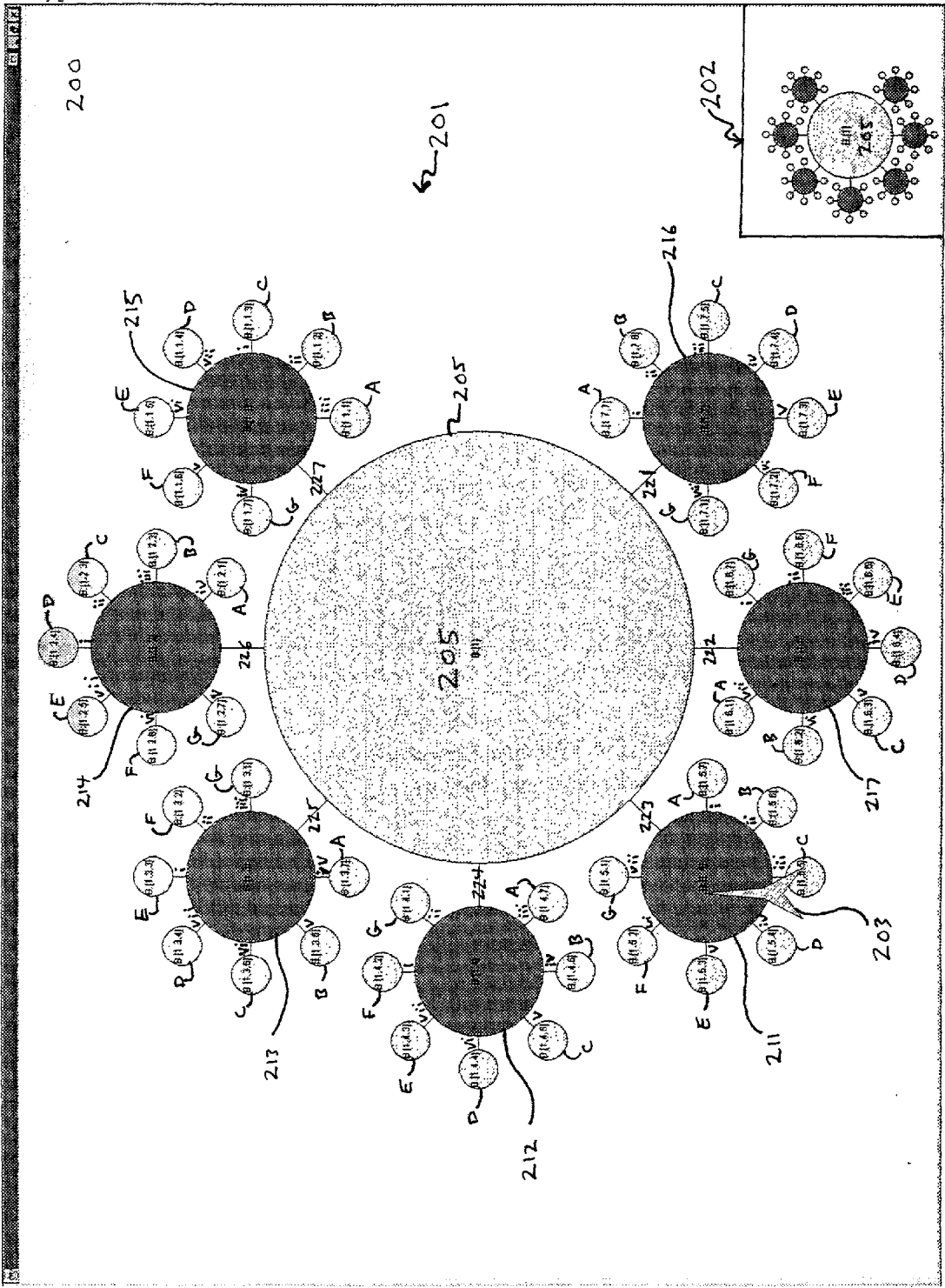
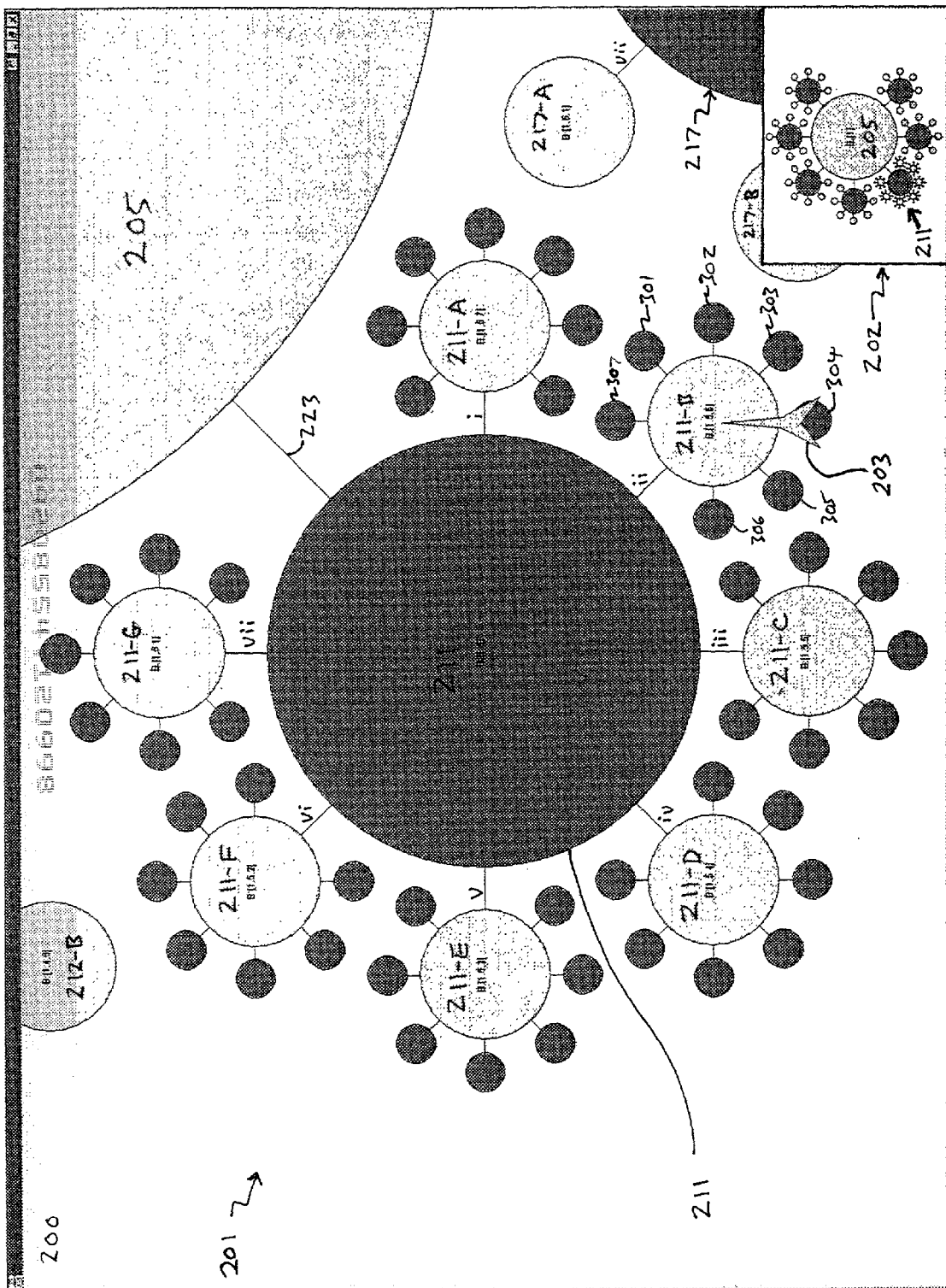


FIG. 3



५७५

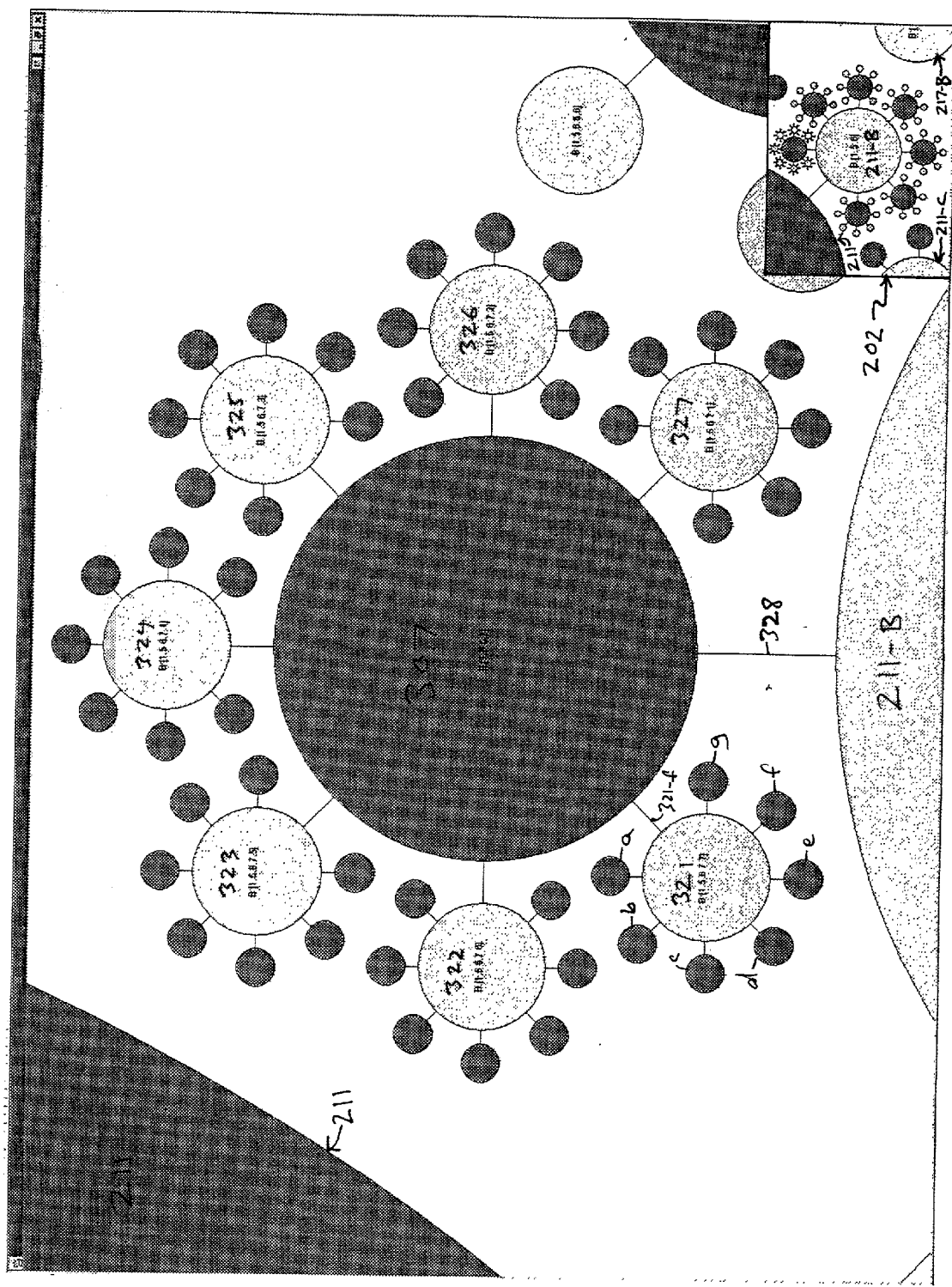


FIG. 6

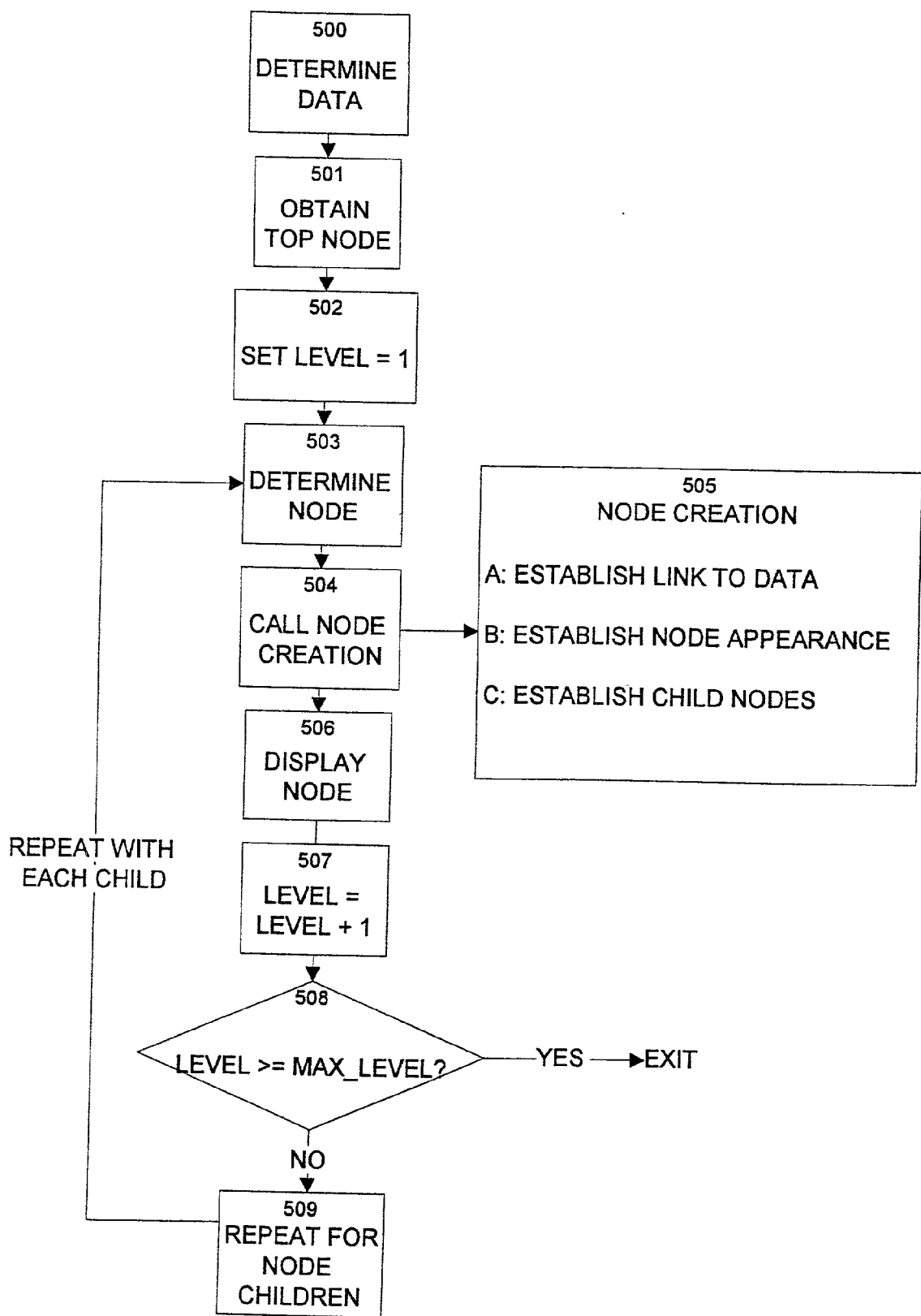


FIG. 7

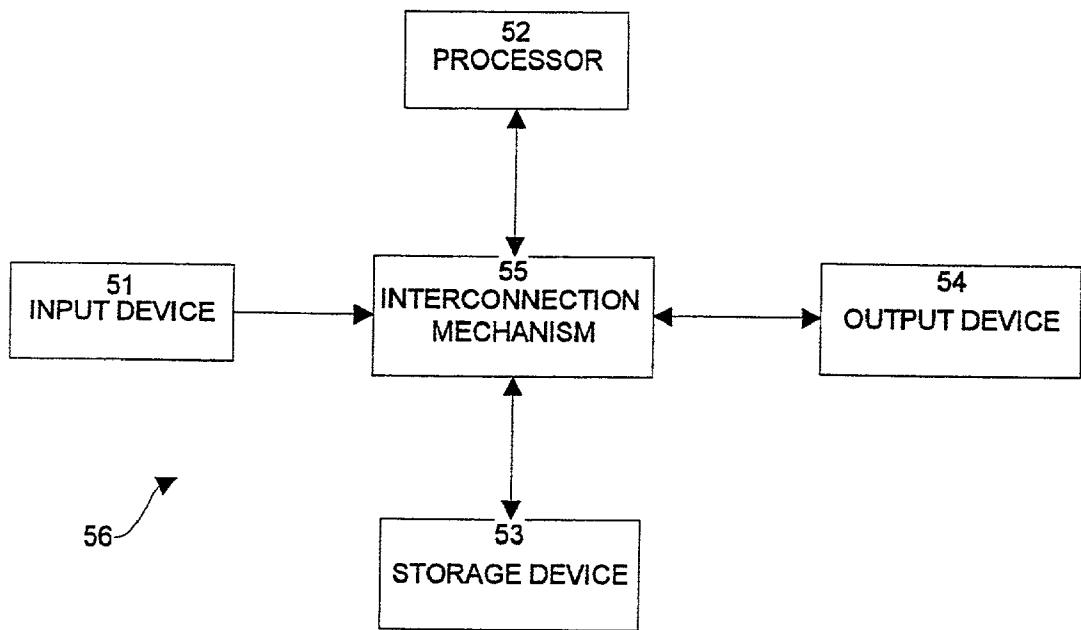


FIG. 8

METHOD AND APPARATUS PROVIDING A GRAPHICAL USER INTERFACE FOR REPRESENTING AND NAVIGATING HIERARCHICAL NETWORKS

FIELD OF THE INVENTION

[0001] The present invention relates to graphical user interfaces, and more specifically, to a computer system employing a software application providing a specialized graphical user interface to efficiently manage, view and navigate relationships between objects having hierarchical relationships with one another, such as those in computer network management applications.

BACKGROUND OF THE INVENTION

[0002] In computer systems, the way in which data is represented to a user is of great importance in many circumstances. Through graphical user interfaces (GUI's), visual representations of data can improve a persons understanding of relationships and characteristics which exist intrinsically within the data. For example, data having fields that have one to one relationships with other fields can be effectively represented using a spreadsheet, with certain fields displayed in a left hand column, and other fields displayed across a top row. Data that is hierarchically related is not well suited for representation within a spreadsheet. Rather, a graphical tree-like structure may be used to represent hierarchical data. The tree depicts a top or parent node for the highest level of data, and has child nodes extending or depending from the parent node. Child nodes represent lower levels in the hierarchy and each successively lower level in the tree is populated with data from successively lower levels in the hierarchy. The overall tree can be displayed in a vertical or horizontal manner with nodes extending outward or downward at each successively lower level, beginning at the top level with the parent node.

[0003] There are many sources of hierarchically organized data, and certain systems are better represented hierarchically than others. Computer networks and individual computer systems generate data which can be effectively represented in a hierarchical manner.

[0004] In the case of a single computer system or data communications device, data stored in a directory on a disk is often hierarchically arranged using folders and sub-directories. Each directory of files and folders represents a separate level in the hierarchy. The hierarchical directory structure of a computer disk provides a convenient way to organize data and files based on characteristics such as file types or on the purpose for which the files and/or data is used.

[0005] Computer networks, on the other hand, typically comprise a combination of computer systems interconnected with a number of network communications devices such as modems, hubs, routers, bridges, switches and so forth. The way in which information is transferred over the computer network is based upon which data communications protocols are in operation and upon how each device is configured or managed. Typically the devices and computers on a single network are owned and managed by an individual entity or enterprise. By way of example, universities or companies typically own, operate, manage and control a network of computers and data communication devices which form a

campus or corporate-wide data communications network. Such networks may be interconnected with other networks, such as those from government agencies, or other entities, to form a worldwide system of interconnected networks, such as the Internet.

[0006] An entire computer network can be broken down logically into various hierarchical levels. For instance, a high speed fiber optic link controlled by high speed routers can form the first level in a hierarchy. This top level network may interconnect many smaller regional networks associated with specific buildings, cities, or geographical areas, each of which represents a second level in the network hierarchy. Each regional network may interconnect specific purpose departmental networks representing a third level in the network hierarchy. Each departmental network may include many individual subnetworks of computers, terminals, printers, file and web servers, and so forth to form the fourth level of the hierarchy, and so forth. Each computer and data communications device may then be considered on an individual basis as the fifth level in the hierarchy and may include various internal devices or peripherals which form a sixth hierarchical level.

[0007] Due to the large amounts of hierarchical data maintained in individual computer systems and computer networks, these systems must be effectively managed or users will quickly begin to lose productivity. Computer systems management refers to the organization, storage and control of data within one or more computing devices. Computer network management refers to the ability to control the operation of computers and associated data communications devices on a network to provide efficient routing of data and network operation. Software applications have been developed for these hierarchical management tasks.

[0008] A typical computer systems management software application presents a graphical user interface to the user which displays information about the computer system. A widely known prior art computer management application called "File Manager", produced by Microsoft Corporation of Redmond Washington, can display hierarchical disk directory structure information in a horizontal tree-like manner. File manager is usually used for simple computer systems management tasks such as organizing files and directories. File Manager examines the file structure on a disk and displays directories and files which are at the same level in the hierarchy as small folder and file icons arranged in columns associated with that level. If files and/or sub-folders exist within other folders, File Manager displays file and sub-folder icons underneath and indented to the right of the parent folder icon in which those files and sub-folders exist. The icons are displayed as a tree using interconnected lines which form a path back to the uppermost folder in the directory. The indentation and positioning of file and folder icons provides a visualization of the hierarchical nature of the file system structure for the portion which is graphically displayed.

[0009] In the case of computers networks, network management software applications are used to display and control the data communication devices that form the network. Network management software executes on a management workstation coupled to the network and allows a network administrator to remotely control the operation of

the network devices. Remote operation and management of devices is provided by standardized protocols such as the Simple Network Management Protocol (SNMP) and the Remote Monitoring (RMON) protocol. SNMP, for example, provides device agents which execute on each remotely located network device under management control. The device agents communicate with the network management software application. Each SNMP device agent can collect data related to the operation of the device and can report problems back to the management application. The management application can control each device, via instructions sent to the device agent. For instance, if a router in a network is experiencing congestion due to high data traffic, the agent in that router may be instructed by the network manager, using the network management application, to re-route certain data to an underutilized segment of the network. SNMP data is stored in a Management Information Base, or MIB, which can be hierarchically arranged.

[0010] Current network management applications present the structure of the managed network in a graphical tree-like manner, much like the computer system management applications (i.e., File Manager) discussed above. Essentially, the management software displays a graphical representation of the physical network as long lines with devices located at various locations. The image of a large computer network can grow to become quite lengthy, having hundreds, or even thousands of devices extending therefrom. The network manager can graphically navigate around the network by scrolling the image of the network left or right on the display using, for example, a mouse pointer to select left/right and up/down scroll bars. When an icon is displayed representing a network device of interest, the network manager can select this device. Device selection causes data associated with that device to be displayed. The data is typically device statistic data obtained from the SNMP MIB for that device.

SUMMARY OF THE INVENTION

[0011] Various problems are present in the ways in which prior art systems display and navigate through hierarchically arranged data. Since the data is represented more or less as a tree-like structure, only certain portions of the tree are displayable at any one time. For instance, in both the computer network management application and in the File Manager application noted above, only localized portions of a large hierarchy of data are visible to the user. In File Manager, if a folder icon contains many sub-files and folders, and this folder is selected, the display is re-drawn to show (i.e., to make visible) all of the sub-folders and files within the selected folder. In the network management case, a network manager often spends much of his or her time simply navigating around the image in order to arrive at the location of the device desired.

[0012] The problem with this is that once the desired object (i.e., networked device or file or other data) is arrived at, the user can no longer see a general representation of the folders, files, network devices, network structure or hierarchical data at the same level or above the parent folder, other than the data immediately surrounding the object displayed. That is, since the interface displays an extended file listing or a large network segment as a relatively flat object, the object and its neighboring objects are rarely visible at the same time. This is because the tree-like display or an extended graphical representation of a network segment

only provides a limited amount of space for displaying various levels of hierarchical data, and there is little scalability to different levels of detail. In other words, once more than a few heavily populated folders or network segments are expanded via user selection, the user quickly becomes disoriented with respect to what area or location of the file system, network or other hierarchical data representation they are currently viewing.

[0013] Another problem with tree-like interfaces used to display hierarchical data is that navigation becomes difficult. Once a user is viewing a detailed list of the contents of one level of the hierarchy, for example, a large network segment, it is quite often the case that scrolling is required within the list of items (i.e., files or network nodes for example) in order to make visible the item of interest at that level or on that segment. Once the parent folder or graphical object (i.e., icon) containing the lower level items is scrolled off the display, the user must re-scroll back to where the parent folder is visible in order to navigate up to the next level in the hierarchy by re-selecting the parent object. Alternatively, the user must condense one or more icons representing neighboring devices or folders, for example, in order to "make room" on the display for one or more devices of interest. Still further, to, for example, delve "down" into a hierarchy having many levels, the user must select a device of interest at each level and "enter" the level of that device, then find and select the next device at that level, and select and "enter" the next level, and so on and so forth. In general, this problem requires many redundant selection, scrolling, finding, selection, scrolling, finding and deselecting operations just to find a particular item within the hierarchy of interest. Once found, the user oftentimes does not recall where they are currently in the hierarchy, and must return "up" to higher levels to gain a sense of direction within the graphical user interface.

[0014] The aforementioned problems combine to have a pronounced effect when using prior art hierarchical interfaces to perform tasks such as moving and copying objects from one level to another. For instance, if a file located at a lower level in the hierarchy is to be copied to another location elsewhere in the hierarchy, a typical interface supports the drag-and-drop concept. In drag-and-drop, the user should be able to find a file, select it, and while maintaining it in a selected state (i.e., with a mouse pointer), the user should be able to drag the file icon to another location and release selection of the item. The interface will interpret this action as an intent to copy the file or item to that location. However, in tree-like hierarchical interfaces, this becomes difficult due to the requirements of having to scroll from one location to another in the hierarchy in order to find the source and destination for an item. It is frequently the case that the source and destination cannot be arranged so as to both be visible on the screen at once. In such a case, an alternative means of copying the item must be used.

[0015] Similar problems exist for network management applications which attempt to display a network hierarchy as a tree-like structure. The network manager must scroll through a potentially lengthy graphical display of network communication lines in order to arrive at a device of interest. After selecting the device to view data associated with the device, and then reverting to the graphical view, the user may not recall the whereabouts of the current location within the network.

[0016] The present invention avoids these problems with prior art interfaces designed to represent hierarchical data. The invention provides a unique interface and navigation mechanism within the interface to effectively represent a large number of hierarchical levels while at the same time presenting the user with a sense of location within the hierarchy.

[0017] More specifically, the invention provides a computer system comprising at least one input device, an output device, a memory device, and a processing unit coupled to the input device, the output device and the memory device. The computer system in a preferred embodiment is a network management workstation. The processing unit executes a computer program stored in the memory device. Preferably, the computer program is a network management software application, such as the CiscoWorks2000 network management software manufactured by Cisco Systems, Inc. of San Jose, Calif. The computer program controls the output device to display a graphical user interface and controls the input device(s) to accept input to manipulate a view of objects displayed within the graphical user interface. The graphical user interface displays objects derived from hierarchically arranged data, wherein each object displayed is graphically scaled in size and proportion directly to the level of the hierarchy to which it relates and is positioned in relation to other objects within the hierarchy of the data. Due to the arrangement of objects in the GUI, the invention overcomes the aforementioned problems of prior art systems related to navigation and conceptualization of where the user is within the hierarchy of data represented.

[0018] Preferably, one or more child objects directly related one level below a parent object in the hierarchy will be positioned radially adjacent to and equidistantly from the parent object. The child nodes can be spaced equidistant from each other as well. The child objects are scaled smaller than the parent object from which they depend in the hierarchy thus indicating their presence on a lower hierarchical level. All objects on a level can be similarly colored, but can be differently colored from objects at different levels, thus further distinguishing hierarchical levels.

[0019] An embodiment of the interface described herein can be used, for example, by a network management graphical user interface monitoring program for use in computer network management, such as the CiscoView application manufactured by Cisco Systems, Inc. of San Jose, Calif. CiscoView provides a graphical user interface representing real time network data and devices in order to allow user to easily understand large volumes of network management data. The interface described herein can be used in such an application.

[0020] Also within the computer system, each object displayed on the graphical user interface is depicted graphically as being interconnected to other objects to which it is directly related to in a hierarchy of the data, so as to form a graphical representation of the hierarchical nature of each object as it relates to other objects. This interconnection represents data links in a computer network, for example. Also within the computer system, an object hierarchically related directly above another object is a parent object and is displayed larger, and an object hierarchically related directly below another object is a child object and is displayed adjacent to and smaller than the parent object. With

these definitions, the input device may be used to select a child object on the graphical user interface and, in response, the computer program will generate a zoom-in view of that child object including any child objects related to that child object. This allows a user to easily navigate up and down and across levels within the hierarchy.

[0021] To zoom-in to a level, the graphical user interface displays a first view of individually selectable hierarchically arranged objects including a parent object and a plurality of child objects peripherally arranged about the parent object. Then, upon a zoom-in selection of a child object with the input device, the processing unit executes the computer program to transform the display of the graphical user interface to a zoomed-in view in which the parent object is relocated and proportionally increased in size to appear as a grand-parent object and the child object selected is relocated and proportionally increased in size to appear as a centrally located parent object. Furthermore, previously undisplayed child objects that are hierarchically related below the child object selected become displayed and peripherally arranged about the centrally located parent object. In this manner, each time the user changes levels in the hierarchy, new levels are exposed.

[0022] The system also provides a zoom-out capability wherein upon a zoom-out selection of the grand parent object with the input device, the processing unit executes the computer program to transform the display of the graphical user interface to a zoomed-out view. In the zoom-out view, child objects disappear from the graphical user interface and the first parent object is proportionally reduced in size to appear as a child object and the grand-parent object is relocated and proportionally reduced in size to appear as a centrally located parent object and the child object and other previously undisplayed child objects that are hierarchically related below the centrally located parent object become displayed and peripherally arranged about the centrally located parent object. Essentially, zoom-out is the reverse of zoom-in and allows the user to navigate up levels in the hierarchy.

[0023] The graphical user interface further includes a locale map providing a user with an indication of where in the hierarchy the objects exist that are currently on display in the graphical user interface. The locale map allows the user to move up and down any number of levels that are displayed in the locale map merely by selecting the node at the level desired. The GUI is then redrawn with the selected node as the centrally located node and displays child nodes, and any parent nodes, as noted above.

[0024] By providing an interface of this sort, any type of hierarchical data can be represented and explored in a fast and efficient manner without getting lost in the hierarchy.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular description of preferred embodiments of the invention, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

[0026] FIG. 1 illustrates a top-level view of a typical prior art computer network.

[0027] FIG. 2 illustrates a more detailed view of a particular segment of the prior art network shown in FIG. 1.

[0028] FIG. 3 is a top level view of the hierarchical interface configured according to the invention.

[0029] FIG. 4 is a view of a second level of the hierarchy for a selected node from the top level view of FIG. 3.

[0030] FIG. 5 is a view of a third level of the hierarchy for a selected node from the second level view of FIG. 4.

[0031] FIG. 6 a view of a fourth level of the hierarchy for a selected node from the third level view of FIG. 5.

[0032] FIG. 7 is a flow chart of the processing steps involved to create, scale and position nodes on the graphical user interface based upon hierarchical data.

[0033] FIG. 8 illustrates the architecture of a computing system which implements the processing according to the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0034] FIG. 1 illustrates a high level view of the architecture of a typical managed computer network 100. FIG. 1 is used to illustrate the concepts of hierarchical components and how a computer network can be viewed and represented as such. The invention, however, is not limited to being applied only to represent a hierarchical computer network architecture such as that of FIG. 1, even though preferred embodiments of the invention are well suited for this purpose. As will be explained in more detail later, the invention can be used to represent other forms of hierarchical data as well.

[0035] In FIG. 1, managed computer network 100 includes backbone network 110 which interconnects regional networks 111 through 117. Backbone network 110 is formed from high speed routers 110-a through 110-g interconnected with high speed data links 118. Each regional network 111 through 114 is formed from sets of routers labeled "a", "b", "c" and so forth within each regional network 111 through 117. Managed computer network 100 represents, for example, a corporate-wide computer network in which backbone network 110 spans a large geographical area such as an entire state and each regional network 111 through 117 represents the corporate computer network facilities within a specific region such as a city in that state.

[0036] FIG. 2 illustrates a more detailed view of the regional network 111 illustrated in FIG. 1. Regional network 111 is formed from the interconnection of seven data communications hubs 111-a through 111-g. When regional network 111 is illustrated at this level of detail, it can be seen that the hubs 111-a through 111-g link departmental networks 130 through 136 to the regional network 111. Departmental networks 130 through 136 are formed from interconnected routers labeled "a", "b", "c" and so forth on each departmental network 130 through 136. Each departmental network 130 through 136 represents, for example, the computer network associated with a single department within the company owning the entire network 100. Extending from each departmental network 130 through 136 are several

subnetworks 137 through 154. Connected to each subnetwork 137 through 154 are various computers, printers, file servers, network devices, and other computing systems referred to herein as networked hosts 155 through 183. During the operation of computer network 100, networked hosts 155 through 183 communicate packets, cells, frames, and other data between one another via the network infrastructure according to various data communications protocols.

[0037] In this example computer network 100, high speed routers 110-a through 110-g, hubs 111-a through 111-g, and regional network routers 130 through 136 can be controlled remotely via a network management application. The network management application is a software program which executes on a network management workstation. The network management workstation is represented, for example, by host 168 coupled to subnetwork 151 on departmental network 132 which extends from regional network 111. The departmental network 132 may be the systems administration department within the company that owns and operates network 100.

[0038] A network manager (i.e., a user) can use the network management software from host 168 to remotely control the operation and data flow between hosts and data communication devices on the network 100. Protocols such as the Simple Network Management Protocol (SNMP) and the Remote Monitoring (RMON) protocol can be used for this purpose. To assist the network manager in this task, the network management software application presents a graphical interpretation of the computer network 100 in the form of a graphical user interface (GUI). An example of such an interface in accordance with the parent invention is illustrated in FIGS. 3 through 6.

[0039] FIGS. 3 through 6 illustrate a graphical user interface configured according to a preferred embodiment of the invention. Absent the labeling of the elements in the figures provided for purposes of this description, each of FIGS. 3 through 6 is a representation of an actual screenshot captured from a computer system executing a program embodying the invention.

[0040] In FIG. 3, display 200 depicts a graphical user interface (GUI) 201 configured according to a preferred embodiment of the invention. GUI 201 represents hierarchical data in a graphical manner which quickly conveys to the user the hierarchical relationships which exist between different elements of the data. The hierarchy depicted by GUI 201 in FIG. 3, as will be explained in detail below, can be quickly grasped by a user viewing the display 200. The graphical hierarchy is accomplished through the use of scaling, size, placement and color or shading of the various elements that represent data and their interrelationships.

[0041] The GUI 201 includes a top node 205 and, for example, seven child nodes 211 through 217 symmetrically arranged in orbit around the centrally located top node 205. Each of the child nodes 211 through 217 is hierarchically related one level below the top node 205 in the data hierarchy, as conveyed graphically by the interface 201. The scaling and size of each child node 211 through 217, as compared to the top node 205, provides an indication of the hierarchical relationship between each node. Data links 221 through 227 indicate that the child nodes 211 through 217 directly relate from the top node 205 in the data hierarchy.

That is, child nodes **211** through **217** are smaller than top node **205** and node **211** is coupled to top or central node **205** via a data link **223** which indicates that it is on a level in the hierarchy below that of top node **205**. In other words the child nodes directly relate or depend from a top or central node **205**.

[0042] In a similar manner, each child node **211** through **217** has, for example, seven respective child nodes A through G which are hierarchically related below child nodes **211** through **217**. Again, the lower hierarchical relationship is indicated by the smaller size and scale of nodes A through G, as well as by the interconnection of data links “j” through “vii” which connect “back up” to nodes **211** through **217**. In general, as shown in **FIG. 3**, top or central node **205** represents the top of the hierarchy, nodes **211** through **217** represent the second level of a hierarchy, and nodes A through G extending “down” from nodes **211** through **217** represent the third level in the hierarchy.

[0043] The view of GUI **201** in **FIG. 3** can represent any type of hierarchical or stored data. In a preferred embodiment, each node (top node **205**, child nodes **211** through **217**, and “grand-child” nodes A through G) represents data related to elements of a computer network and GUI **201** is an interface employed by a network management application. An example of a network management application for which the technology provided by GUI **201** is designed is the CiscoWorks2000 network management software application produced by Cisco Systems, Inc. of San Jose, Calif.

[0044] As a more detailed example of how the GUI **201** represents a computer network, suppose the example computer network **100** illustrated in **FIGS. 1 and 2** is represented by the GUI **201** in **FIG. 3**. In this case, the top node **205** represents and corresponds to the backbone network **110**, while the child nodes **211** through **217** represent and correspond to the regional networks **111** through **117**, respectively. Moreover, the nodes A through G of each of the child nodes **211** through **217** correspond to the departmental networks of regional networks in computer network **100**. By way of example, regional network **111** in **FIG. 2** is represented by child node **211** in the GUI **201** in **FIG. 3** and child nodes **211-A** through **211-G** in **FIG. 3** represent departmental network links **111-a** through **111-g** in **FIG. 2**, respectively.

[0045] Before further description of the operation of GUI **201** is given, another element of GUI **201** will be briefly explained. In **FIG. 3**, GUI **201** also includes a map **202** located in the lower right-hand corner of the display **200**. Map **202** provides the user viewing display **200** with a conceptual view of where the user is located at any one moment in time within the overall hierarchy shown by the GUI **201**. That is, map **202** provides an aerial or “forest through the trees” view of sorts. In **FIG. 3**, map **202** displays the same information as the main portion of GUI **201**, since the view in **FIG. 3** is the top level view of the hierarchy of data represented. As will be explained, when a user selects a node in GUI **201**, computational aspects of the invention change the level of the GUI **201** to a more or less detailed view of the selected node. In essence, the user can zoom in and/or zoom out to different levels in the hierarchy represented in GUI **201** by simply selecting a desired node to be displayed as the center node (e.g., top node **204** in **FIG. 3**) on the display **200**. As will be explained, the map **202**

provides a sense of where the user is within the hierarchy of GUI **201**, aside from the view provided by the main portion (i.e., all but lower right hand corner) of GUI **201** as previously explained.

[0046] Upon start-up of a network management application configured with the GUI **201** according to this invention, the user is presented with the display **200** as it appears in **FIG. 3**. That is, preferably, the network management application presents a start-up view showing the top level of a network hierarchy. As previously explained, GUI **201** is presently configured in this example to show three levels of the network hierarchy which correspond to the backbone network **110**, regional networks **111** through **117**, and departmental networks A through G. More often than not, the user will want to navigate through the network hierarchy beyond (or below) the first three levels shown in **FIG. 3**.

[0047] To navigate using GUI **201**, the user merely positions pointer **203** onto a desired node within GUI **201** and selects that node via a selection mechanism. A preferred embodiment of the computer system (i.e., network management station) of this invention employs a hand operated selection mechanism such as mouse or trackpad which controls the graphical pointer **203** on display **200**. To select a node, pointer **203** is positioned over child node **211**, for example, as illustrated in **FIG. 3**. In this example, selecting node **211** represents a decision by the user to see further details of regional network **111** (**FIG. 2**). Upon selection of child node **211**, for example, by pressing one of two mouse buttons, the user is presented with a brief animated zoom-in process (to be explained) which results in the display **200** being formatted as shown in **FIG. 4**.

[0048] In **FIG. 4**, a close up or zoomed-in view of level two of the hierarchy represented by the GUI **201** is illustrated with a focus on node **211**. Top node **205** is still visible as the large semi-circle in the upper right corner of display **200**. However, the center of display **200** is now dominated by a more detailed view of node **211**, which was selected by the user via pointer **203** in **FIG. 3**. Furthermore, the grand-child nodes **211-A** through **211-G** (i.e., grand-child nodes of top node **205**) are now displayed as child nodes of node **211**. Nodes **211-A** through **211-G** are displayed larger and are shown in more detail.

[0049] The GUI **201** in **FIG. 4** also now displays the child nodes of nodes **211-A** through **211-G** which were previously not displayed in **FIG. 3**. Examples of child nodes of nodes **211-A** through **211-G** are nodes **301** through **307**, each of which is directly hierarchically related below (i.e., is a child node of) node **211-B**. Nodes **301** through **307** are child nodes of node **211-B** and are grand-child nodes of node **211** and are great-grand-child nodes of top node **205**. Note that only child nodes **301** through **307** are labeled in **FIG. 4**, and that the other nodes **211-A** and **211-C** through **211-G** also have similarly positioned and related child nodes which have not been labeled.

[0050] In the network management example, the view presented in **FIG. 4** corresponds to the view of computer network **100** in **FIG. 2**. Thus, node **211** represents regional network **111** and each node **211-A** through **211-G** corresponds to a respective one of departmental networks **131** through **137** in **FIG. 2**. As a specific example, node **211-B** may represent departmental network **131** and nodes **301** through **307** which depend therefrom can represent subnet-

works 144 through 150. In this manner, a network manager can navigate through a symbolic representation of network 100 via GUI 201.

[0051] Note that not all nodes in FIG. 4 are labeled in order to avoid confusion in the figure and to avoid drawing space limitations. It should also be noted that FIG. 4 may illustrate more nodes than are illustrated in the network 100 in FIG. 2. This is because FIG. 2 is given by way of example only, and is not an actual representation derived from a real network. However, the nodes in FIG. 4 and other figures are actually created based upon data that exists for elements of a hierarchical data structure.

[0052] Due to the hierarchical nature of FIG. 4, and other figures to be discussed, it is to be understood by the reader that nodes of similar size and hierarchical placement are on a similar hierarchical level as one another, and that not all nodes will be labeled in each figure. For example, the unlabeled nodes which directly depend from node 211-C are shown in the figure as being directly hierarchically related below node 211-C.

[0053] The display 200 of GUI 201 in FIG. 4 provides numerous indications to a user (e.g., a network manager) who is viewing the display 200 as to where that user "is" in relation to the hierarchy being represented. Top node 205 remains somewhat visible and data link 223 clearly shows the hierarchical relationship between top node 205 and its child node 211. Moreover, neighboring nodes 217 and its child nodes 217-A and 217-B are partially visible as well. The positioning of these neighboring nodes is made apparent during the animated transition that takes place from FIG. 3 to FIG. 4 upon selection of node 211 by the user using pointer 203.

[0054] While it is difficult to convey the animated nature of the transition from FIG. 3 to FIG. 4 using just these two figures alone, suffice it to say that the start of the operation is shown by FIG. 3. Then, upon selection of node 211 in FIG. 3, the network application employing the technology of this interface presents a brief series of more and more focused images, each of which shows a more zoomed in view of node 211 than the previous image, until the final image of display 200 is presented as it appears in FIG. 4. The entire animation process takes approximately one second, and displays approximately twenty or thirty separate images ranging in scale from the top zoomed-out or distant view in FIG. 3 to the zoomed-in or close-up view in FIG. 4. Each image is successively displayed as a sequence of graphical frames and provides the impression or the appearance that the user is getting closer and closer and is zooming-in to the selected node 211. In this manner, the user is presented with an animated zoom-in effect for the transition between FIGS. 3 and 4.

[0055] Another important characteristic of GUI 201 which is illustrated at the hierarchical level shown in FIG. 4 is the presentation of map 202. In FIG. 4, map 202 now indicates how node 211, which is the central node of display 200 in FIG. 4, fits into the higher level(s) of the hierarchy. As shown in map 202, top node 205 is shown, as well as child node 211. In map 202, each child node that depends from top node 205 is illustrated as having a number of child nodes. However, a distinction of child node 211 in map 202 in FIG. 4 is that node 211 (in the map 202) not only has its direct child nodes visible, but also has its grand-child nodes visible

as well whereas other child nodes 212 through 217 do not. The user can quickly glance at map 202 to identify which node within the overall hierarchy is displayed on the central area of display 200. This due to the fact that only child node 211 in map 202 is drawn to the same level of detail as the larger image of node 211 and its children and grandchildren, as presented in the central portion of display 200.

[0056] Once the user has quickly grasped the information related to the hierarchy as presented in FIG. 4, the user can delve deeper into the hierarchy using the same process as previously described to effect the transition from FIG. 3 to FIG. 4. That is, in FIG. 4, as the user views the various nodes presented at various level by GUI 201, the user can decide to select another child (or a child's child) node, such as indicated by the positioning of pointer 203 over node 211-B. Upon selection of node 211-B, a similar zoom-in effect is carried out which results in the display 200 presenting GUI 201 as it appears in FIG. 5.

[0057] FIG. 5 shows the close-up or zoomed-in view of centrally positioned node 211-B after it has been selected in FIG. 4. In FIG. 5, parent node 211, which represents regional network 111, is visible in the upper left hand corner of display 200, and child nodes 301 through 307, which represent the subnetworks networks 144 through 150 (FIG. 2) are graphically drawn to a larger scale than they were in FIG. 4. Also shown in FIG. 5 are the child nodes of nodes 301 through 307. These nodes were not visible in FIG. 4. For example, node 307 has seven child nodes 321 through 327 which depend from node 307 in the hierarchy of network data represented by the nodes. Child nodes 321 through 327 represent computer equipment and devices within subnetwork 145. For example, nodes 321 through 325 may represent networked hosts 163 through 167. Other child nodes of nodes 301 through 306 are visible in FIG. 5, but are not labeled as their identity is clear from their depiction with GUI 201.

[0058] In FIG. 5, the map 202 also has been updated to provide the user with a sense of which node is being viewed as the central node on display 200. That is, map 202 displays the same information as the central part of display 200 within the map 202, and further provides a representation of higher level nodes in the hierarchy such as node 211, and top node 205. This allows a user to glance at the map 202 to gain a sense of position within the hierarchy displayed by GUI 201.

[0059] Again, should the user desire to view information relating to data elements and objects that are located lower in the hierarchy than the objects, elements or nodes displayed in FIG. 5, the user may again select a node which will become the central node in another view. As an example, in FIG. 5, the user can position pointer 203 on node 307 and select this node. Node 307 represents, for example, one of the departmental networks such as departmental network 145 from computer network 100 illustrated in FIG. 2. The transition process described above for displaying a zoomed-in view of a hierarchy level is repeated upon selection of node 307 and the user is presented with the GUI 201 displayed in FIG. 6.

[0060] FIG. 6 illustrates another view of another level of detail in the hierarchy represented by GUI 201. In FIG. 6, node 307, as selected from FIG. 5, is now the central node in the display 200. Node 307 represents, for example,

subnetwork **145** which is illustrated in **FIG. 2**. The child nodes **321** through **327** are visible in larger scale and in more detail, and the child nodes of node **321** through **327** are also now in view. Likewise, map **202** is updated to indicate to the user where the current view of GUI **201** exists in the hierarchy of data represented.

[0061] As explained, the interface provided by embodiments of the invention provides a convenient mechanism to display hierarchical data. In a preferred embodiment, nodes from the examples given correspond to various elements or objects in a computer network environment. The nodes provide a wealth of information regarding the computer network element or object which they represent. Thus, in **FIG. 6** for example, child nodes **321** through **327** represent actual computer or data communication equipment attached to subnetwork **145** in **FIG. 2**. For example managed host **163** may be represented by node **321** in **FIGS. 5 and 6**. As such, child nodes **321-a** through **321-g** represent, for example, various system or device data related to host **321**.

[0062] It is to be understood that the example nodes within GUI **201** in **FIGS. 3 through 6** are graphically created based upon mapped hierarchical data that is provided by the computer network management software. The data may be gathered, for instance, by the SNMP protocol which uses the computer network **100** to remotely query actual data communications and computer devices on the network **100**. The devices respond with status information regarding many types of information.

[0063] For example, in **FIG. 2**, suppose networked computer device **163** on subnetwork **145** is a file server which serves files to other computer systems (i.e., devices **163**, **164**, **165** and **167**) within subnetwork **145**. Each node **321-a** through **321-g**, may represent a disk subsystem on the file server. If file server **163** is equipped with an SNMP agent, it can be remotely controlled and queried for information regarding the status of each disk node **321-a** through **321-g** by the network management workstation host **168**, which is located across the regional network on departmental network **132** within subnetwork **151**. The remote queries and control commands are provided by the network management software which incorporates the GUI **201**. The SNMP protocol maintains a hierarchical database which is accessible to the network management workstation **168**. If file server **163** suddenly loses access to one of its disks such as **321-d**, the SNMP agent on file server **163** can signal this condition via an exception to the remotely located network management application on host **168**. Upon receipt of this error condition, the network management software application can, for instance, alter the color of the node **321-g** which represents the failed disk, as well as the color of node **321** which represents the file server **163** in GUI **201**. The color alteration indicates that there is a problem with file server.

[0064] In the event that the network manager is viewing the top level of the network (e.g., **FIG. 3**) at the time the error occurs, the network management application configured according to this invention can propagate the error condition up to the top level of the hierarchy (i.e., display the error in **FIG. 3**). This is accomplished, for instance, by changing all data links (i.e. data links **321f**, **328**, **300**, **211-ii**, and **223**) leading to the lower level node **163** to indicate an error condition. The data links can be highlighted or can begin to flash on and off, for example. When the network

administrator notices a data link that begins to flash to indicate an error, such as data link **223** in **FIG. 2**, the network administrator can “drill down” into the hierarchy via the aforementioned zoom-in node selection process. At each level, the network administrator can select the node that is linked to the next higher node in the hierarchy having a flashing or highlighted data link. Once the network administrator reaches a view such as that of **FIG. 5**, the node **321** which represents the device with error can be indicated in red, for example. The network manager can then directly select this node, which will cause node **321** to be the centrally located node on display **200**. The administrator can then analyze data displayed within the node on the GUI **201** to determine the nature of the error with disk **321-d**. The network management application can then use SNMP to control the file server **163** to either disable the faulted disk corresponding to **321-d**, or to hot swap in a new disk, for example.

[0065] In another example, network management applications such as CiscoView and CiscoWorks2000 allow network managers to graphically view a network in real time. Real time network viewing means that devices can become active and inactive on the fly and can generate exceptions and SNMP conditions that must be handled appropriately. In a case where a device is using the Remote Monitoring (RMON) protocol to report its status, the network management application can set thresholds of error reporting for certain RMON situations. The color of nodes on GUI **201** can be altered accordingly, based on the RMON error and on threshold settings in effect. Through the use of the GUI **201** described herein, the network manager can reduce management overhead by selectively choosing to inquire only into those errors that propagate up the hierarchy to a visible node. The error level can be indicated by color. This can improve network troubleshooting capabilities and the GUI **201** can allow a network manager to quickly arrive at the location of the error, without having to scroll through lots of network segments and select a device or object at each hierarchical level. Instead, the invention allows the manager to skip many levels and head straight to the device with the problem. Moreover, the map **202** on the interface keeps the manager abreast of where he is in the network.

[0066] The aforementioned examples illustrate some of the usefulness of the functionality of GUI **201** configured according to the invention. Once hierarchical data is gathered for any purpose, the GUI **201** can be used to represent this data as noted above. The applicability of GUI **201** to network management applications is particularly advantageous in situations such as that described.

[0067] An important point to note with respect to GUI **201** is that not only does the interface provide the ability to zoom-in and drill down to low levels of hierarchical data, but the interface provides the ability to also zoom back out to upper levels in the hierarchy. In this case, the processes presented to the user is the exact opposite of zooming in.

[0068] In a preferred embodiment, to zoom out, the user can perform one of two steps. First, the user can merely select a node that appears anywhere on the GUI **201**, including nodes within the map **202**, that is above the present centrally located node on GUI **201**. In such a case, the interface processing will perform an animated zoom-out process that progressively redraws the GUI **201** from the

zoomed-in current view to the view of the selected node as being the centrally located node in GUI 201. By way of example, suppose the user is viewing GUI 201 as it appears in FIG. 6. If the user selects node 211 with pointer 203 from either the upper left hand corner of display 200, or from within the map 202, the GUI 201 will then scale all nodes and present a GUI that appears like that in FIG. 4, in which node 211 is the centrally located node. In this example, it is noted that the user has effectively jumped up or zoomed-out two levels in the hierarchy. That is, the user, by selecting node 211 from FIG. 6, has chosen to move up two levels and has requested to view the hierarchy from the level represented by node 211.

[0069] Note that moving up or zooming out and moving down or zooming in multiple levels can be performed in all instances, as long as the user can see a node somewhere on GUI 201, including nodes map 202, to which the user would like to zoom to. Thus three, four or more levels can be skipped in either the up or down hierarchical direction merely by selecting a visible node at the desired level.

[0070] The second way in which the user can zoom in or out is by using separate zoom-in and zoom-out buttons provided on the selection input device, such as a computer mouse. Most input devices such as mice are provided with computer systems and include at least two mouse buttons. In an alternative embodiment, the left button can be used to zoom-in one level, while the right button can be used to zoom-out one level. Thus, if pointer 203 is over any node within GUI 201, depending upon which button is depressed, the GUI 201 will either provide zoom-in or zoom-out processing to the next lower or higher hierarchical level.

[0071] The way in which the GUI 201 determines how to display, position, scale, and size each node will now be discussed in detail. Before an analysis of each step required to produce GUI 201 is provided, principles of the calculations involved as part of the invention will be provided in order to fully comprehend the processing steps involved.

[0072] The processing provided by the invention to create and display the nodes uses fractal geometry. A reference to the principles of fractal geometry is provided in "Fractal Geometry of Nature" by Benoit Mandelbrot, published in 1998 by WH Freeman & Co., ISBN 0716711869. The scaling of each node is controlled by a self-avoiding scaling size and placement schema.

[0073] In a preferred embodiment, this scaling, size and placement schema is based on the following principles. For any given node, the distance from the node's center to one of its child's center is called "s". The radius of a node on display 200 is denoted by "c", and a scaling factor that is used to create node sizes is denoted by "r". When comparing the size and scale of a parent node, in relation to one of its child nodes, the child node's radius is $r * c$. Furthermore, as one goes from a child node to the child's node child node (i.e., a grand child of the parent node), the distance between the centers of the child node and the child's child node is $r * s$. For a node to have equally spaced N child nodes, the scaling factor r is given by the formula:

$$r = \frac{1}{1 + 1/\sin(\pi/N)}$$

[0074] and the optimal radius c is given by, and can be no bigger than:

$$c_n < \frac{s_n}{r + 1}$$

[0075] for $n=1, 2, 3, \dots Q$, where Q is the maximum number of child levels to be displayed on GUI 201.

[0076] Using the above formulas, the nodes when displayed are guaranteed to be self-avoiding at all possible scales.

[0077] It is noted that for node peers (i.e., child neighbors) at the same hierarchical level, the connection between two of such nodes must be extended beyond twice the node family radius L, such that:

$$L = \sum_{n=1}^{\infty} s_1 * r^{n-1} = s_1 / 1 - r$$

[0078] That is, L is the radius of a parent node and all of its child nodes. The radius L thus originates from the center of the parent of a family of nodes, and extends to the perimeters of the furthest child. L is useful in situation where more than one parent or top node exists and are linked to each other in a non-hierarchical relationship. That is, there may be two node on the same level that are linked to each other.

[0079] In the above formulas, the root or top node always has a radius for calculation purposes of 1, and child nodes all have smaller radii. Using this notion, L can be used to properly computed the length of a connection formed between two parent nodes, once the number of children of each parent is known.

[0080] A preferred embodiment of GUI processing comprises a program written in an object-oriented programming language, such as Java or C++. When implemented in Java, the entire GUI 201 can be embodied within a Java applet. Using an object oriented language allows a class to be defined that describes all of the information related to a node. The code developed which created the example screens depicted in FIGS. 3 through 6 refers to each node as a bubble. Information such as the radius of the bubble, its position and relation to other parent bubbles in the hierarchy, how many and which child bubbles the bubble has, and what icon or graphical shape is to represent the bubble are all provided in the class describing each bubble.

[0081] FIG. 7 illustrates a flow chart of the processing steps used in a preferred embodiment to create and graphically size, scale and position nodes that represent hierarchical data within GUI 201. In step 500, the GUI processing accepts the hierarchical data as a series of node data structures or objects. Hierarchical data may be obtained from a

database, supplied over a network, generated at runtime, or otherwise provided by or to the application program providing the processing shown in FIG. 7. In a preferred embodiment, the program embodying the steps of FIG. 7 is a network management program and the hierarchical data relates to networked computing devices obtained via SNMP or another such network analysis or control protocol.

[0082] Once the hierarchical data is received in step 500, step 501 determines the root or top node of the data. Step 502 then sets a counter called LEVEL equal to a value of 1. LEVEL is used to control how many levels of the hierarchy are displayed on GUI 201, and has a maximum attainable value. The number of levels displayed can be preset or determined by a user via a selected parameter. Step 503 obtains the node selected in step 501 and determines which node is to be created. During the first execution of step 503, the only known node is the top node defined in step 501 from the data obtained in step 500. Step 504 then calls a node creation routine 505.

[0083] Node creation routine 505 first establishes a link to the data for the current node in step 505-a, and then establishes the current node's appearance in step 505-b. Step 505-b uses the fractal geometry to compute how the node will be drawn in the GUI 201. Step 505-b computes the radius for the node being created, as well as a scaling factor and also computes the positioning of the node, as it relates to other nodes represented by data in the hierarchy. Since the current node on the first pass of step 505 is the top node, the radius, as noted above, is always scaled to a value of 1, and the placement the top node, as the parent node on the screen, is always centered in the graphical user interface (i.e. 201 in FIGS. 3 through 6).

[0084] Each graphical object related to a node of data, as well as actual nodes of data (i.e., from the database), both contain information concerning what parent node that node or data depends from in the hierarchy, and each parent node data includes a list of all child nodes which depend from that node of data and hence from the graphical object. Once the current nodes appearance within GUI 201 has been determined, step 505-c then establishes the child node objects for the current node, as determined by examining the data link established via step 505-a.

[0085] Positioning or screen placement of each child node can be accomplished, in a preferred embodiment, in the following manner. Since the number of child nodes for any parent node is known from the data link obtained in step 505-a, the processing divides three hundred sixty degrees by the number of child nodes which will depend from a parent. The result is the angle of separation required for each child node to be displayed in a circumferential manner around a parent node. This allows each child node to be established in step 505-C with a parameter allowing even spacing of each child about the parent node. If only one child node exists, it can be placed at a user selectable location, as defined by parameter setting upon which GUI processing is based.

[0086] Once an object for each child node of the current node is established in memory, (but is not yet rendered on GUI 201), processing returns to step 506 where the current node fully created by steps 505-a through 505-c is displayed on GUI 201. If step 506 detects that a parent node of the node just displayed exists on the screen, step 506 then draws the interconnection from the parent node to the node just

rendered. Each interconnection is a graphical object in and of itself and represents a data link between two nodes. To render each interconnection of data link, the coordinate center of the parent node and the coordinate center of the child node are used to draw a straight line between the two center points. In this manner, each data link appears graphically to interconnect the parent node to each of its child nodes.

[0087] Once the node is displayed in step 506, step 507 increments the LEVEL count by one, to indicate that a new level of the hierarchy has been rendered on GUI 201. In step 508, if LEVEL is greater than or equal to a maximum level value MAX_LEVEL, the current thread of processing finishes and exits. When all threads have displayed nodes up to the maximum level, each thread exits and the display processing of FIG. 7 is complete.

[0088] If the maximum number of hierarchical levels has not been displayed on GUI 201 step 508 allows step 509 to call processing step 503 repeatedly as a separate processing thread for each child node established as an object in memory in step 505-c. As such, step 503 effectively is a recursive entry point into the processing which gets passed a child object to treat as a new object from which other children may depend. In this manner, the children of each child node object get established, and each child object then gets created by the processing of steps 505-a through 505-C and the child node gets displayed in step 506. Once the LEVEL counter reaches the maximum levels for the thread of each child processing routine, processing recursively exits that thread and no more nodes are established or displayed.

[0089] It is to be understood that the processing provided beginning at the top node can effectively be used beginning at any other node, as provided by a user selecting a node via pointer 203 in GUI 201. Thus, if a user selects a low level node in the hierarchy with a mouse, the processing treats the selected node as the top level node at step 501 for purposes of rendering any child nodes. The same is true for zoom-out processing. In this case, an upper level node that is selected is treated as the central or top node for purposes of the processing of FIG. 7. In this manner, the graphical nodes are displayed as explained above.

[0090] Note that the map 202 as previously described is merely treated as a smaller screen in the low corner of display 200. Thus all drawing and scaling are the same as for the main part of display 202, except that the scaling is much smaller.

[0091] It is to be understood that the invention contemplates not only application of the general technology provided by GUI 201 as explained herein but also specific application of this technology to uses such as network management and navigation, and file system management, and other applications requiring such an interface and hierarchical navigation system between interrelated hierarchical data elements.

[0092] As previously noted, embodiments of the invention may be implemented on a computer system. An example architecture of such a computer system is illustrated in FIG. 8. In FIG. 8, the computer system 56 comprises intercoupled components 51 through 55. The computer system 56 generally includes an interconnection mechanism 55

coupling an input device **51**, a processor **52**, a storage device **53** and an output device **54**. In a preferred embodiment of this invention, the computer system **56** is the network management station previously described.

[0093] The input device **51** receives data in the form of commands, computer programs or data files such as hierarchical database files and other information as input to the computer system **56** from users or other input sources. Typical examples of input devices include a keyboard, a mouse, data sensors, and a network interface connected to a network to receive another computer system's output.

[0094] The interconnection mechanism **55** allows data and processing control signals to be exchanged between the various components **51** through **54** of the computer system **56**. Common examples of an interconnection mechanism are a data bus, circuitry, and in the case of a distributed computer system, a network or communication link between each of the components **51** through **54** of computer system **56**. The computer network **100** is an example of an interconnection mechanism between different computers.

[0095] The storage device **53** stores data to be processed by the computer system **56**. Typical storage devices may include computer memory and non-volatile memory such as hard disks, optical disks, or file servers and database servers locally attached to the computer system **56** or accessible over a computer network.

[0096] The processor **52** executes computer programs loaded into the computer system **56** from the input or storage devices. Typical examples of processors are Intel's Pentium, Pentium II, and the 80x86 series of microprocessors; Sun Microsystems's SPARC series of workstation processors; as well as dedicated application specific integrated circuits (ASIC's). There may be many processors or just a single processor. The processor **52** may also be any other microprocessor used in computing devices for performing information processing.

[0097] The output device **54** is used to output information from the computer system **56**. Typical output devices may be computer monitors, displays, LCD screens or printers, speakers or recording devices, or network connections linking the computer system **56** to other computers. A computer device such as that shown in **FIG. 8** commonly has multiple input, output and storage devices as well as multiple processors.

[0098] Generally, in operation, the computer system **56** shown in **FIG. 8** is controlled by an operating system. Typical examples of operating systems are MS-DOS, Windows95, Windows98 and Windows NT from Microsoft Corporation, or Solaris and SunOS from Sun Microsystems, Inc. These terms may be trademarks of their respective companies. As the computer system **56** operates, input such as text data, text file or Web page data, programs, commands, and database data including hierarchical data, received from users or other processing systems, may be temporarily stored on storage device **53**. Certain commands cause the processor **52** to retrieve and execute stored programs, such as a program implementing the hierarchical data representation processes of the invention as discussed above.

[0099] The programs executing on the processor **52** may obtain more data from the same or a different input device, such as a network connection providing data from another

computing device on a computer network. The programs may also access hierarchical data in a database or file for example, and commands and other input data may cause the processor **52** to begin to perform operations on the input data. Data may be generated which is sent to the output device **54** to be either displayed, such as GUI **201**, saved, or further processed. The output data may be held for transmission to another computer system or device for further processing by other application programs as well.

[0100] Typical examples of the computer system **56** are personal computers and workstations, hand-held computers, dedicated computers designed for a specific purposes, and large main frame computers suited for use by many users. The invention is not limited to being implemented on any specific type of computer system or data processing device, nor is it limited to a single processing device or type of computer network.

[0101] It is noted that the invention may be implemented in hardware or circuitry which embodies the logic and processing disclosed herein, or alternatively, the invention may be implemented in software in the form of a self contained software package or application, or other type of program stored on a computer readable medium like a CD-ROM disk or floppy disk(s), such as the storage device **53** shown in **FIG. 8**. In the later case, the invention in the form of computer program logic and executable instructions is read and executed by the processor **52** and instructs the computer system **56** to perform the functionality disclosed as the invention.

[0102] If the invention is embodied as a computer program or as software on a disk, the computer program logic is not limited to being implemented in any specific programming language. The invention in software may be written in an object oriented language, or may be written in a structured or unstructured programming language. For example, commonly used programming languages such as C, C++, JAVA, FORTH, Visual BASIC, as well as others, such as list processing languages may be used to implement the logic and functionality of the invention. Furthermore, the subject matter of the invention is not limited to implementation on currently existing computer processing devices or programming languages, but rather, is meant to be able to be implemented in many different types of environments in both hardware and software which exists now and that will exist in the future.

[0103] Furthermore, combinations of embodiments of the invention may be divided into specific functions and implemented on different individual computer processing devices, processors and/or systems which may be interconnected to communicate and interact with each other over a network or a data bus, for example. Dividing up the functionality of the invention between several different computers is meant to be covered within the scope of the invention as well.

[0104] A preferred embodiment of the invention is documented in Appendix A as source code written in the JAVA programming language. All copyright in the source code listed in Appendix A is reserved by its appropriate owner(s). It is a violation of copyright law to duplicate this code, other than for patent distribution purposes. It is to be understood that unauthorized creation, duplication, or encoding of the JAVA code listed in Appendix A onto a computer readable medium, such as an optical or magnetic storage medium

such as a computer disk, also constitutes patent infringement of the invention defined in this patent, and as covered by one or more of the claims herein. It is also to be understood that

unauthorized execution of the code in Appendix A by a computer processing apparatus constitutes patent infringement of one or more claims of this patent.

APPENDIX A

```
/******Angle.java*****  
*/
```

```
package inc.awt.soap;
```

```
/**  
    The angle is represented as a sin-cos (or basis vector) pair.  
    This will facilitate fast calculations of lines, etc.<p>
```

Note that $ax^{**2} + ay^{**2}$ must always equal 1.

```
    @see SoapBubbles  
*/
```

```
public class Angle extends FloatingPair  
{  
    {  
        x = 1.0;  
        y = 0.0;  
    }
```

```
    Angle() { }
```

```
    Angle(double rad)  
    {  
        x = Math.sin(rad);  
        y = Math.cos(rad);
```

```

    }

    Angle(double sin, double cos)
    {
        x = sin;
        y = cos;
    }

    double getRadian()
    {
        double r = Math.acos(y);
        // if the sin is negative, that means
        // that the angle is on the other side of
        // the circle and we must subtract r from 2Pi to adjust.
        if (x < 0.0)
            r = 2.0 * Math.PI - r;

        return r;
    }
}

/*****Bubble.java*****/
*/
package inc.awt.soap;

import java.awt.*;
import java.util.*;
import inc.util.*;

/**
    Classes that implement Bubble are instantiated by SoapBubbles.
    Your class is then passed a reference object that you may use
    to instantiate yourself appropriately. <p>

    Classes that implement the Bubble interface must be derived from
    java.awt.Component or java.awt.Container so that it can be used
    as a lightweight component in SoapBubble's Panel.<p>

    @see java.awt.Container
    @see java.awt.Component
*/
public interface Bubble
{
    /**
        Initialize this bubble (establish identity).<p>
        The gui aspect of this object is not gaurenteed at this time.
        All gui-related determinations should be made only during
        paint()/update() times.<p>

        @param laba Logical Abstract Bubble Address
        @param reference special reference object.
    */

```

```

void init(Laba laba, Object reference);

/**
    Shut down yourself (go away). Will normally be used to flush out
    unused (non-visible or off-screen) objects.<p>

    It is possible that in the future, this will be replaced by
    a serialization mechanism.<p>
*/
void dispose();
public void setBounds(Rectangle r); // do not implement -- derived in Component of implementor
public Rectangle getBounds(); // do not implement -- derived in Component of implementor
/**
    Get children classes we need to instantiate.<p>

    Please keep like classes/objects adjacent so that we SoapBubbles
    can group them, if need be. Classes are checked for object
    equivalence ( == ) so use the same instance of Class (the Class object, not
    the actual instantiation of the object that Class refers to!) for the
    same class.<p>

    @return An array of Bubble interfaces or Class objects is returned. Class
    objects must be classes that implements the Bubble interface.
*/
Object[] queryChildren();

/**
    Get indices of where each distinct group begins. <p>

    @return an array of indices to where each logical grouping
    begins in the array returned by queryChildren().
    Can return null, in which case SoapBubbles will do
    its own grouping.
*/
int[] queryChildGroups();

/**
    The paint method.
*/
public void paint(Graphics g, Rectangle r);

/**
    The update method, as normally implemented by Components.
*/
public void update(Graphics g);

/**
    Events to be handed off to the bubble for processing
*/
public void processBubbleEvent(AWTEvent e);
}

```

```

/*****BubbleEvent.java*****/
*/
package inc.awt.soap;

import java.awt.event.*;
import java.util.*;
//import inc.util.*;

/**
 * Event generated by SoapBubbles.
 */

public class BubbleEvent extends EventObject
    implements java.io.Serializable
{
    BubbleEvent(SoapBubbles source)
    {
        super(source);
    }
}

/*****Circle.java*****/
*/
package inc.awt.soap;

/**
 * This represents a circle to SoapBubbles.
 * @see SoapBubbles
 */

public class Circle extends FloatingPair
{
    /**
     * Radius of circle
     */
    public double r = 1.0;

    Circle() { }
    Circle(double x, double y, double r)
    {
        this.x = x;
        this.y = y;
        this.r = r;
    }
}

/*****FloatingPair.java*****/
*/
```

```
package inc.awt.soap;

public abstract class FloatingPair
    implements Cloneable, java.io.Serializable
{
    public double x = 0.0;
    public double y = 0.0;
}

/*****FloatingRectangle.java*****/
*/
package inc.awt.soap;

public abstract class FloatingRectangle
    implements Cloneable, java.io.Serializable
{
    public double x1 = 0.0;
    public double y1 = 0.0;
    public double x2 = 4.0;
    public double y2 = 4.0;
}

/*****FPoint.java*****/
*/
package inc.awt.soap;

class FPoint extends FloatingPair
{
}

/*****FRect.java*****/
*/
package inc.awt.soap;

public class FRect extends FloatingRectangle
{
    /** Create an interpolated FRect based on <code> 0 <= t <= 1 </code>.<p>
    */
    public FRect interpolate(FRect er, double t)
    {
        FRect ir = new FRect();

        ir.x1 = x1 + (er.x1 - x1) * t;
        ir.x2 = x2 + (er.x2 - x2) * t;
        ir.y1 = y1 + (er.y1 - y1) * t;
        ir.y2 = y2 + (er.y2 - y2) * t;

        return ir;
    }
}
```

```
    }  
}  
  
/*****GenericBubble.java*****/  
*/  
package inc.awt.soap;  
  
import java.awt.*;  
import java.util.*;  
  
/** Generic Bubble Class  
*/  
  
public class GenericBubble extends Canvas implements Bubble  
{  
    Class[] children = new Class[12];  
    Laba laba = null;  
  
    private Color[] color = { Color.red, Color.green, Color.blue, Color.orange };  
  
    public void init(Laba laba, Object reference)  
    {  
        this.laba = laba;  
        for (int i = 0; i < children.length; ++i)  
            children[i] = getClass();  
    }  
  
    public void dispose()  
    {  
        children = null; // just to be sure this class is disposed!  
    }  
  
    public Object[] queryChildren()  
    {  
        return children;  
    }  
  
    public int[] queryChildGroups()  
    {  
        return null;  
    }  
  
    public void processBubbleEvent(AWTEvent e)  
    {  
        processEvent(e);  
    }  
  
    /** paint a simple filled circle based on our size */  
    public void paint(Graphics g, Rectangle r)  
    {  
        //System.out.println("paint(g, "+r+") of " + this.getClass().getName() + " called");  
        g.setColor(color[laba.getTuple() % color.length]); g.fillOval(r.x+1, r.y+1, r.width-1, r.height-1);  
    }  
}
```

```

        g.setColor(Color.black); g.drawOval(r.x, r.y, r.width, r.height);
        String str = "B:" + laba;
        int w = g.getFontMetrics().stringWidth(str);
        if (w <= r.width)
        {
            int x = (r.x + r.width / 2) - w / 2;
            int y = r.y + r.height / 2;
            g.drawString(str, x, y);
        }
    }

    public void update(Graphics g)
    {
        // don't call super.update()! We're not attached to anything!
        paint(g);
    }
}

```

```

/*****Laba.java*****/
*/
package inc.awt.soap;

/** Logical Abstract Bubble Address
    Tuple indices start at one, as is the mathematical convention. */
public class Laba implements java.io.Serializable, Cloneable
{
    int[] addr;

    public Laba(int tuple)
    {
        addr = new int[tuple];
    }

    /**
        Create a Laba with one more extra tuple than current.
    */
    public Laba(Laba l, int index)
    {
        this(l.addr.length + 1);
        System.arraycopy(l.addr, 0, addr, 0, l.addr.length);
        addr[addr.length - 1] = index;
    }

    /** Clone */
    private Laba(Laba l)
    {
        this(l.addr.length);
        System.arraycopy(l.addr, 0, addr, 0, l.addr.length);
    }

    /** get the index of given tuple */

```

```

    public int getIndex(int t) { return addr[t - 1]; }

    /** Get the index of last tuple */
    public int getIndex() { return addr[getTuple() - 1]; }
    public void setIndex(int t, int index) { addr[--t] = index; }
    public int getTuple() { return addr.length; }

    protected Object clone() throws CloneNotSupportedException
    {
        return new Laba(this);
    }

    public String toString()
    {
        StringBuffer sb = new StringBuffer();

        sb.append("[");
        for (int i = 0; i < addr.length; ++i)
            sb.append("'" + addr[i]
                .append(i < addr.length - 1 ? " : " : "");
        sb.append("]");

        return sb.toString();
    }
}

/*****SoapBubbles.java - Main GUI Pprogram*****/
*/
package inc.awt.soap;

import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.util.*;
import inc.awt.IncFrame;
import inc.util.*;

/**
    Creates 'fractal' 'bubble' or 'node' network representations of objects and their
    heirarchical relationships. <p>

    Child 'Bubble' objects are created on the fly, as needed for display.
    The parent objects specifiy an array of Class types and number to
    represent its children, which will be created and given an instantiation
    object with with to set itself up for. <p>

    Bubbles have an upper max of radial children (usually 8, but may be more). If the parent
    bubble requires more than this, SoapBubbles automatically creates
    intermediate group objects, as necessary. <p>

    Users can click on a bubble and
    automatically that bubble moves to the center of

```



```

.      /      1
.      /      1 - r
.      n = 1
</pre>

```

For peering among children, it is suggested that the peers themselves be treated like children anyway, so as to not cause overlap problems.<p>

Conversely, since these equations are based on carrying *n* to infinity, there is actually more 'room' around children in the real sense, which might be exploited.<p>

NOTES<p>

The root soap bubble is taken to be a radius of 1. All of the smaller bubbles will be of smaller radii.<p>

@author Fred Mitchell, Cisco Systems.

@version 1.0

@see Bubble

@see BubbleEvent

*/

```

public class SoapBubbles extends Container
    implements Runnable,
        java.io.Serializable

```

```

{
    ///
    class AnimData
    {
        View v;
        FRect newr;
        FRect oldr;
        Graphics g;
        Soap newSoap;

        void clear()
        {
            newSoap = null;
            g = null;
            v = null;
        }
    }
}

```

/// Management

Object viewLock = new Object(); // our own internal synchronization for views

AnimData animLock = new AnimData(); // animation lock/sync + data

/// Data

int spokes = 7; // number of spokes per bubble (note that one spoke is for parent)

final int visible_children = 2; // number of levels o visible children on mainline bubbles

```

double r; // scaling factor - calculated by readySoap()
double s; // size spoke length of root soap - calculated by readySoap()
final double s_scale = 1.15; // used by readySoap() to compute s
Circle c = new Circle(); // radius and location of root circle
Angle ang = new Angle(); // Angle of orientation of root circle
FRect focus; // current focus rectangle
FRect newFocus; // newly-selected focus

```

```

Object reference = null; // Reference object that is passed ot bubbles
Class root = null; // Startup or root bubble

```

```

/** Top-level soap bubble */
Soap rootSoap = null;

```

```

/** Soap that has the caret (has focus, or is in the center of screen) */
Soap caretSoap = null;

```

```

//// Inner Classes

```

```

/** This embodies SoapBubbles' view of the individual Bubble.
    @param b associated bubble
    @param level level from the top of heirarchy.
*/

```

```

class Soap implements java.io.Serializable

```

```

{
    /** Root level soap */
    Soap(Bubble b)
    {
        laba = new Laba(1);
        bubble = b;
        laba.setIndex(1, 1);
        bubble.init(laba, reference);
    }

    /** Create a child soap bubble.<p>
        @param parent parent soap
        @param b associated bubble
    */
    Soap(Soap parent, Bubble b)
    {
        //this(b, p.getLaba().getTuple() + 1);
        children[0] = parent;
        bubble = b;
        int i;
        for (i = 1; i < parent.children.length; ++i)
            if (parent.children[i] == null)
            {
                parent.children[i] = this;
                break;
            }
        laba = new Laba(parent.laba, i);
    }
}

```

```

/** free up resources, this Soap is being discarded. */
void dispose()
{
    for (int i = 1; i < children.length; ++i)
        if (children[i] != null)
        {
            children[i].dispose();
            children[i] = null;
        }
    bubble.dispose();
    children = null;
}

//int n; // level (first is 1) get this from laba.size()
double s; // spoke length

Circle circle = null; // position and radius
Angle angle = null; // Angle of the position 0

boolean areWeAGroup = false;
boolean initCalled = false;
boolean disposeCalled = false;
/** Position 0 is reserved for a parent reference.<p>
    If at top level (level 0), the the array MAY be one
    greater than normal. */
Soap[] children = new Soap[spokes + 1];

Bubble bubble;
Laba laba; // address of this bubble

Image icon;

public Laba getLaba() { return laba; }

public View lastView = null; // bookkeeping - last view to resize us.

/** */
public void setSpokeLength(double s) { this.s = s; }
public double getSpokeLength() { return s; }

/** */
public void setCircle(Circle pos) { circle = pos; }
public Circle getCircle() { return circle; }

/** */
public void setAngle(Angle ang) { angle = ang; }
public Angle getAngle() { return angle; }

/** get our associated bubble */
public Bubble getBubble() { return bubble; }

public ProxyBubble createProxyBubble()
{

```

```
        return new ProxyBubble();
    }

    /** Class to use in views that accesses Bubble */
    public class ProxyBubble extends Component implements Bubble
    {
        { // set up bubble to receive events!
            enableEvents( AWTEvent.ACTION_EVENT_MASK
                | AWTEvent.MOUSE_EVENT_MASK
                //| AWTEvent.MOUSE_MOTION_EVENT_MASK
                | AWTEvent.ACTION_EVENT_MASK
                | AWTEvent.KEY_EVENT_MASK
                | AWTEvent.FOCUS_EVENT_MASK
                | AWTEvent.TEXT_EVENT_MASK
                | AWTEvent.ITEM_EVENT_MASK);
        }

        public void init(Laba laba, Object reference)
        {
            if (!initCalled)
                bubble.init(laba, reference);

            initCalled = true;
        }

        public void dispose()
        {
            if (!disposeCalled)
                bubble.dispose();

            disposeCalled = true;
        }

        public void setBounds(Rectangle r)
        {
            //System.out.println("setBounds(" +r+ ") for " + this);
            super.setBounds(r);
            bubble.setBounds(r); // should we be calling this???
        }

        public Object[] queryChildren() { return bubble.queryChildren(); }

        public int[] queryChildGroups() { return bubble.queryChildGroups(); }

        public void paint(Graphics g, Rectangle r)
        {
            System.out.println("paint(g, r) of " + this.getClass().getName() + " called");
            bubble.paint(g, r);
        }

        public void paint(Graphics g)
        {
            System.out.println("paint(g) of " + this.getClass().getName() + " called");
        }
    }
}
```

```

public void update(Graphics g) { bubble.update(g); }
public void processBubbleEvent(AWTEvent e) { bubble.processBubbleEvent(e); }

Soap getSoap() { return Soap.this; }

protected void processEvent(AWTEvent e)
{
    //System.out.println("processEvent " + e);
    super.processEvent(e);
    processBubbleEvent(e);
}

protected void processMouseEvent(MouseEvent e)
{
    super.processMouseEvent(e);

    if (e.getID() == MouseEvent.MOUSE_PRESSED)
        animateToNewSoap(getSoap());
}

/** get number of children (first child index starts at 1) */
public int getSoaps() { return children.length - 1; }

/** Is there a child Soap currently instantiated?
    Use this function to avoid auto-instantiation
    of child soaps. */
public boolean isSoap(int index)
{
    return children[index] != null;
}

/** get child or parent Soap. Parent is at index 0. */
public Soap getSoap(int index) throws InstantiationException, IllegalAccessException
{
    if (children[index] == null && index > 0) // 0 index refers back to the parent Soap. Can only
    be null at the root.
    {
        // create the soap!
        Object o = bubble.queryChildren()[index];
        Bubble b = null; // child bubble
        if (o instanceof Class)
        {
            b = (Bubble) ((Class)o).newInstance();
            b.init(new Laba(laba, index), reference);
        }
        else
            b = (Bubble) o; // the actual bubble (will throw an exception here if wrong type!)

        children[index] = new Soap(this, b);
    }
    return children[index];
}

```

```

/** Get rid of children. Exclude child at exclude. */
public void pruneChildren(int level, int exclude)
{
    for (int i = 1; i < children.length; ++i)
        if (i != exclude && children[i] != null)
        {
            if (level == 0)
            {
                children[i].dispose();
                children[i] = null;
            }
            else
                children[i].pruneChildren(level - 1);
        }
}

/** Prune all children at given level (this == level 0) */
public void pruneChildren(int level) { pruneChildren(level, 0); }

/** Prune all children. */
public void pruneChildren() { pruneChildren(0, 0); }

/** This method exists solely for convenience.
    If you need to override, override getSoap() instead. */
final public Soap getParent() throws InstantiationException, IllegalAccessException { return
getSoap(0); }

/**
public void paint(Graphics g)
{
    Rectangle re = getBounds();
    g.setColor(Color.black);
    g.drawOval(0, 0, re.width, re.height);
} /**/

/** LAYOUT of immediate children.
    Note that this does not cause the children to layout its own
    children!!! <p>

    @param recurse recursion factor, > 0 will make
        children draw children with recurse - 1<p>
*/
void layoutChildren(int recurse) throws InstantiationException, IllegalAccessException
{
    double a0 = angle.getRadian();
    double ar = 2.0 * Math.PI / children.length;
    for (int i = 1; i < children.length; ++i)
    {
        Angle ai = new Angle(ar * i + a0);
        Soap child = getSoap(i);

        child.setAngle(new Angle(Math.PI + ar * i + a0));
    }
}

```

```

        child.setSpokeLength(r * s);
        child.setCircle(new Circle(circle.x + s * ai.x, circle.y + s * ai.y, circle.r * r));
        if (recurse > 0)
            child.layoutChildren(recurse - 1);
    }
}
}

/**
Layout Manager for SoapBubble.
This lays out the views.<p>

STRINGS:
<pre>
Main    - central mapping
NW      - View is a bird's eye - place it in the NW corner
NE      - View is a bird's eye - place it in the NE corner
SW      - View is a bird's eye - place it in the SW corner
SE      - View is a bird's eye - place it in the SE corner
</pre>
*/
class BubbleLayout implements LayoutManager
{
    Hashtable list = new Hashtable(); // note that the components are used as KEYS, the strings
used as VALUES!!
    final int bird = 5; // bird's eye divisor

    public void addLayoutComponent(String name, Component comp)
    {
        list.put(comp, name); // comp is SUPPOSED to be the key!
    }

    /** get named view */
    View getView(String name)
    {
        for (Enumeration e = list.keys(); e.hasMoreElements(); )
        {
            View v = (View) e.nextElement();
            if (name.compareTo((String) list.get(v)) == 0)
                return v;
        }

        return null;
    }

    /** Components added MUST implement the Bubble interface!
    This will throw an exception if Component is not right!*/
    public void removeLayoutComponent(Component comp)
    {
        list.remove(comp);
    }

    public Dimension minimumLayoutSize(Container parent) { return new Dimension(100, 50); }

```

```
    public Dimension preferredLayoutSize(Container parent) { return new Dimension(650, 250);  
}  
    public void layoutContainer(Container parent)  
    {  
        try  
        {  
            for (Enumeration e = list.keys(); e.hasMoreElements(); )  
            {  
                View v = (View) e.nextElement();  
                String s = (String) list.get(v);  
                Dimension d = parent.getSize();  
  
                if ("Main".compareTo(s) == 0)  
                {  
                    v.setBounds(0, 0, d.width, d.height);  
                    v.setUcs(caretSoap);  
                }  
  
                else if ("NW".compareTo(s) == 0)  
                {  
                    v.setBounds(0, 0, d.width / bird, d.height / bird);  
                    v.setUcs(caretSoap.getParent());  
                }  
  
                else if ("NE".compareTo(s) == 0)  
                {  
                    v.setBounds(d.width - d.width / bird, 0, d.width / bird, d.height / bird);  
                    v.setUcs(caretSoap.getParent());  
                }  
  
                else if ("SW".compareTo(s) == 0)  
                {  
                    v.setBounds(0, d.height - d.height / bird, d.width / bird, d.height / bird);  
                    v.setUcs(caretSoap.getParent());  
                }  
  
                else if ("SE".compareTo(s) == 0)  
                {  
                    v.setBounds(d.width - d.width / bird, d.height - d.height / bird,  
                                d.width / bird, d.height / bird);  
                    v.setUcs(caretSoap.getParent());  
                }  
            }  
        }  
        catch (IllegalAccessException iae)  
        {  
            log.dialog(iae);  
        }  
        catch (InstantiationException ie)  
        {  
            log.dialog(ie);  
        }  
    }  
}
```

```

    }

    /** View component for the indicated class.
     * The bubbles are laied out in Uiversal Coordiate Space. You need
     * Views on that coordinate space to map the object to a viewable
     * region. <p>
     *
     * There can be multiple views, and for the purposes of animation,
     * begin and end views will be created. Also, a 'bird's eye' view
     * may be created which will allow a bird's eye viewing area.<p>
     */
    class View extends Container implements java.io.Serializable
    {
        FRect ucs = null; // Universal Coordinate Space region to be viewed.
        Image offScreen = null; // place to do double-buffering.

        View() { setLayout(new ViewLayout()); }

        /**
         * Layout Manager for SoapBubble.
         * We need to be able to get at SoapBubble's parameters. That's why
         * we must be an inner class.
         */
        class ViewLayout implements LayoutManager
        {
            /** Components added MUST implement the Bubble interface!
             * This will throw an exception if Component is not right!*/
            public void addLayoutComponent(String name, Component comp) { Bubble b = (Bubble)
comp; }

            /** Components added MUST implement the Bubble interface!
             * This will throw an exception if Component is not right!*/
            public void removeLayoutComponent(Component comp) { Bubble b = (Bubble) comp; }

            public Dimension minimumLayoutSize(Container parent) { return new Dimension(100, 50);
}
            public Dimension preferredLayoutSize(Container parent) { return new Dimension(650, 250);
}
            public void layoutContainer(Container parent)
            {
                layoutBubbles();
                layoutView((View) parent);
            }

            void layoutView(View v)
            {
                synchronized(viewLock)
                {
                    Component[] proxy = v.getComponents();
                    for (int i = 0; i < proxy.length; ++i)
                    {
                        Rectangle r;

```

```

        Soap.ProxyBubble pb = (Soap.ProxyBubble) proxy[i];
        pb.setBounds(r = v.computeBoundingBox(pb.getSoap()));
    }
}
}

public void update(Graphics g)
{
    super.update(g);
    //paint(g);
}

/** note that this method must not be executed concurrently with any other view! */
public void paint(Graphics g)
{
    System.out.println("View paint for " + this);
    synchronized(viewLock)
    {
        if (needNewOffscreenBuffer())
            offScreen = createImage(getSize().width, getSize().height);

        // Set up and clean buffer
        Graphics goff = offScreen.getGraphics();
        try
        {
            if (goff instanceof Graphics2D)
                ((Graphics2D) goff).setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_SPEED);
        }
        catch (NoClassDefFoundError e) { } // if running under 1.1, catch this error!

        goff.setClip(0, 0, getSize().width, getSize().height);
        goff.clearRect(0, 0, getSize().width, getSize().height);

        setBubblePhysicalCoords();
        renderBubbles(goff);
        g.drawImage(offScreen, 0, 0, Color.white, this);
    }
}

private boolean needNewOffscreenBuffer()
{
    Dimension size = getSize();
    return (offScreen == null ||
        size.width != offScreen.getWidth(this) ||
        size.height != offScreen.getHeight(this));
}

void setBubblePhysicalCoords()
{
    setBubblePhysicalCoords(rootSoap);
}

```

```

void renderBubbles(Graphics g)
{
    renderBubbles(g, rootSoap);
}

/** render this Bubble and all child Bubbles */
void renderBubbles(Graphics g, Soap s)
{
    Bubble b;

    // render children first so spokes may be properly obscured.
    for (int i = 1; i <= s.getSoaps(); ++i)
        if (s.isSoap(i))
            try { renderBubbles(g, s.getSoap(i)); } catch (Exception e) {} // should never throw an
exception!

    // now render ourself.
    if ((b = s.getBubble()) != null)
    {
        Rectangle r = b.getBounds();
        if (g.getClipBounds().intersects(r))
        {
            renderSpoke(g, r, s);
            b.paint(g, r);
        }
    }
}

/** Render the spoke of the child back to parent */
void renderSpoke(Graphics g, Rectangle r, Soap s)
{
    try
    {
        Soap p; // parent
        if ((p = s.getParent()) != null)
        { // draw a line from the center of child rectangle to the center of the parent rectangle
            Rectangle pr = p.getBubble().getBounds();
            g.setColor(Color.black);
            g.drawLine(r.x + r.width / 2, r.y + r.height / 2,
                pr.x + pr.width / 2, pr.y + pr.height / 2);
        }
    }
    catch (Exception e)
    {
    }
}

/** Bounding box for the Soap object for this View */
Rectangle computeBoundingBox(Soap s)
{
    // The bounding box for the Soap object is taken to be the
    // square that encloses the bubble, that is, a square that
    // will be 2r on a side, and will have an upper corner at

```

```

    // (x-r, y-r).

    Circle c = s.getCircle();
    FPoint p = new FPoint();
    FRect box = new FRect();

    // translate central point of circle for this view.
    p.x = c.x - ucs.x1;
    p.y = c.y - ucs.y1;

    // create the box coordinates
    box.x1 = p.x - c.r;
    box.y1 = p.y - c.r;
    box.x2 = p.x + c.r;
    box.y2 = p.y + c.r;

    // Scale to view
    Rectangle rv = getBounds();
    Rectangle r = new Rectangle();
    r.x = (int) (box.x1 * rv.width / (ucs.x2 - ucs.x1));
    r.y = (int) (box.y1 * rv.height / (ucs.y2 - ucs.y1));
    r.width = (int) (2.0 * c.r * rv.width / (ucs.x2 - ucs.x1));
    r.height = (int) (2.0 * c.r * rv.height / (ucs.y2 - ucs.y1));

    return r;
}
/** run through all Bubbles and set their physical coordinates
    @deprecated
    */
void setBubblePhysicalCoords(Soap s)
{
    ((ViewLayout) getLayout()).layoutView(this);
}

/** Set Universal Coordinate System based on given Soap. */
void setUcs(Soap s)
{
    if (s == null)
        s = rootSoap;

    ucs = calcUcs(s);
}

void setUcs(FRect ur) { ucs = ur; }
FRect getUcs() { return ucs; }

/** calculate Universal Coordinates based on given Soap. */
FRect calcUcs(Soap s)
{
    FRect ucs = new FRect();

    // Currently, we set the width or height (whichever is smaller) to
    // be 2 * sum( s.spokeLength() * r ** i, i = 0, visible_children )

```

```

double length;
double rr;
int i;
for (i = 0, rr = 1.0, length = 0.0;
    i <= (visible_children + 1);
    ++i, rr *= r)
    length += s.getSpokeLength() * rr;
//length *= 2.0;

if (getSize().width < getSize().height) // width (x-axis) is the narrower one
{
    ucs.x1 = s.getCircle().x - length;
    ucs.x2 = s.getCircle().x + length;
    double hlength = length * (double) getSize().height / (double) getSize().width;
    ucs.y1 = s.getCircle().y - hlength;
    ucs.y2 = s.getCircle().y + hlength;
}
else // height (y-axis) is the narrower one (or equiv).
{
    ucs.y1 = s.getCircle().y - length;
    ucs.y2 = s.getCircle().y + length;
    double wlength = length * (double) getSize().width / (double) getSize().height;
    ucs.x1 = s.getCircle().x - wlength;
    ucs.x2 = s.getCircle().x + wlength;
}

return ucs;
}

/**
 * Given various Floating-based coords, return with
 * the actual coordinates to be passed on to Graphics.
 * @see Soap
 */
protected Rectangle getPhysical(Circle cir)
{
    Rectangle re = new Rectangle();
    Rectangle b = getBounds();

    // first, convert the circle into rectangle coords.
    double x1 = cir.x - cir.r;
    double x2 = cir.x + cir.r;
    double y1 = cir.y - cir.r;
    double y2 = cir.y + cir.r;

    // next, do the conversion
    re.x = (int) ((x1 - focus.x1) * b.width / (focus.x2 - focus.x1) + b.x);
    re.y = (int) ((y1 - focus.y1) * b.height / (focus.y2 - focus.y1) + b.y);
    re.width = (int) ((x2 - x1) * b.width / (focus.x2 - focus.x1));
    re.height = (int) ((y2 - y1) * b.height / (focus.y2 - focus.y1));

    return re;
}

```

```

        protected Point getPhysical(FPoint fp)
        {
            Point p = new Point();
            Rectangle b = getBounds();

            p.x = (int) ((fp.x - focus.x1) * b.width / (focus.x2 - focus.x1) + b.x);
            p.y = (int) ((fp.y - focus.y1) * b.height / (focus.y2 - focus.y1) + b.y);
            return p;
        }
    }

    static private long threadCount;

    /** for JavaBeans */
    public SoapBubbles()
    {
        setLayout(new BubbleLayout());
        add(new View(), "SE");
        add(new View(), "Main");

        Thread t = new Thread(this, "SoapBubbles-" + ++threadCount);
        t.setDaemon(true);
        t.setPriority(Thread.MIN_PRIORITY);
        t.start();
    }

    /**
     SoapBubbles needs the appropriate starter class,
     and a reference to pass to then upon initialization.

     @param reference Reference object that will be passed to all
        subsequent bubbles.
     @param starter Root bubble class from which all other bubbles stream.
    */
    public SoapBubbles(Object reference, Class starter) throws InstantiationException,
    IllegalAccessException
    {
        this();
        setReference(reference);
        setStarter(starter);
        readySoap();
    }

    /** Set the reference object that will be passed to all Bubble instances. */
    public void setReference(Object reference)
    {
        this.reference = reference;
    }

    /** Set the root Bubble class */
    public void setStarter(Class starter)
    {
        this.root = starter;
    }

```

```

    }

    /**
     * Animation thread.
     */
    public void run()
    {
        synchronized(animLock)
        {
            for(;;)
            {
                try { animLock.wait(); } catch(Exception e) {}
                doAnimation();
            }
        }
    }

    private void doAnimation()
    {
        synchronized(viewLock)
        {
            for (double t = 0.1; t <= 1.0; t += 0.1)
            {
                FRect ir = animLock.older.interpolate(animLock.newer, t);
                animLock.v.setUcs(ir);
                (((View.ViewLayout) animLock.v.getLayout()).layoutView(animLock.v);
                animLock.v.setBubblePhysicalCoords();
                animLock.v.paint(animLock.g);

                try { Thread.sleep(100); } catch (Exception e) {}
            }
        }

        // finalize
        caretSoap = animLock.newSoap;
        try { layoutBubbles(); } catch (Exception x) {}
        validate();
        animLock.clear();
        animLock.notify();
    }

    /** Checks to make sure all necessary setup is in place for the layout.
     * May also do some Soap- pruning, as necessary. */
    void readySoap()throws InstantiationException, IllegalAccessException
    {
        // Set up initial parameters.
        r = 1.0 / (1.0 + 1.0 / Math.sin(Math.PI / spokes));
        s = c.r * (r + 1.0) * s_scale; // s > c(r+1), make s s_scale greater (typically around 1.15)

        // check to make sure the caret is set
        if (caretSoap == null)
        {
            if (rootSoap == null) // do we need to establish the root Soap object?
                rootSoap = new Soap((Bubble) root.newInstance());
        }
    }

```

```

        caretSoap = rootSoap;
        rootSoap.setAngle(ang);
        rootSoap.setCircle(c);
        rootSoap.setSpokeLength(s);
    }

    // More to come, as needed.
}

/** enumerate all the soaps in the system. */
public Enumeration soaps()
{
    return new Enumeration()
    {
        class Ts { Soap s = null; int i = 1; Ts(Soap s) {this.s = s;} }
        Queue q;

        {
            q = new Queue();
            q.push(new Ts(rootSoap));
        }

        public boolean hasMoreElements()
        {
            return !q.isEmpty();
        }

        public Object nextElement()
        {
            try
            {
                while(hasMoreElements())
                {
                    Ts ts = (Ts) q.peek();

                    if (ts.i > ts.s.getSoaps()) // no more children to check!
                    {
                        q.pop();
                        if (hasMoreElements())
                            ((Ts)q.peek()).i++;
                        return ts.s;
                    }
                    else if (ts.s.isSoap(ts.i)) // children to look at!
                        q.push(new Ts(ts.s.getSoap(ts.i)));
                    else // childless entry in array
                    {
                        ts.i++;
                    }
                }
            }
            catch (Exception e)
            {
            }
        }
    }
}

```

```

        return null; // should never get here!
    }
};
}

/**
 * Layout engine for bubbles.<p>
 * Called by the BubbleLayout manager, which is tightly coupled to
 * SoapBubbles.<p>
 *
 * <code> caretSoap </code> is expected to be set to the appropriate Soap that now
 * "has the focus". If null, will be automatically set to the root Soap.<p>
 */
void layoutBubbles()
{
    try
    {
        readySoap();
        lb_blowBubbles();
        lb_setViewOfBubbles();
        lb_pruneBubbles();
    }
    catch (Exception e)
    {
        log.dialog(e);
    }
}

View getMainView()
{
    return ((BubbleLayout) getLayout()).getView("Main");
}

/** Create and/or position bubbles from the root down to the caret.<p>
 * Note that main-line bubbles will always have at least
 * <i>visible_children</i> level of children 'visible'.<p> */
void lb_blowBubbles() throws InstantiationException, IllegalAccessException, QueueException
{
    // We start by making a list of mainline bubbles. We do this
    // by starting at the caret and working our way back to the root.
    Queue q = new Queue();
    Vector mainline = new Vector();
    for (Soap p = caretSoap; p != null; p = p.getParent())
        q.shove(p); // pull() these back out for LIFO operation.

    while(!q.isEmpty())
        mainline.addElement(q.pull());

    // All children are now instantiated. Perform actual 'fractal layout'
    // calculations on this constellation of children down the mainline.
    rootSoap.setCircle(c);
    rootSoap.setAngle(ang);
}

```

```

    rootSoap.setSpokeLength(s);

    // Calculate all children of the mainline
    for (Enumeration e = mainline.elements(); e.hasMoreElements(); )
        ((Soap) e.nextElement()).layoutChildren(visible_children - 1);
}

/** set our proper coordinate view system of the bubbles. */
void lb_setViewOfBubbles()
{
    Component[] cview = getComponents(); // every component should be a view.
    for (int i = 0; i < cview.length; ++i)
    {
        View v = (View) cview[i];
        v.removeAll();
        for (Enumeration e = soaps(); e.hasMoreElements(); )
        {
            Soap s = (Soap) e.nextElement();
            if (s.getBubble() != null)
                v.add(s.createProxyBubble());
        }
    }
}

/** Get rid of extraneous children. */
void lb_pruneBubbles() throws InstantiationException, IllegalAccessException
{
    int exclude = 0;
    for (Soap p = caretSoap; p != null; p = p.getParent())
    {
        p.pruneChildren(visible_children, exclude);
        exclude = p.getLaba().getIndex();
    }
}

/** Create animation to newSoap, then set newSoap as the caret.
    Note that newSoap must be in the hierarchy. <p>
    At this time, we only animate the main view.<p> */
void animateToNewSoap(Soap newSoap)
{
    // set up animation
    synchronized(animLock)
    {
        animLock.v = getMainView();
        animLock.newr = animLock.v.calcUcs(newSoap);
        animLock.older = animLock.v.getUcs();
        animLock.g = getGraphics();
        animLock.newSoap = newSoap;

        //animLock.notify();
        doAnimation();
    }
}

```

```
}

/** Testing for Bubbles */
public static void main(String[] av)
{
    try
    {
        Frame f = new IncFrame()
        {
            { enableEvents(WindowEvent.WINDOW_EVENT_MASK); }

            protected void processWindowEvent(WindowEvent e)
            {
                if (e.getID() == WindowEvent.WINDOW_CLOSING)
                    System.exit(0);
            }
        };

        f.setLayout(new BorderLayout());
        f.setSize(500, 600);
        SoapBubbles sb = new SoapBubbles(null, Class.forName("inc.awt.soap.GenericBubble"));
        f.add(sb, "Center");
        f.setVisible(true);
    }
    catch (Exception e)
    {
        System.out.println("Error: " + e);
    }
}
```

What is claimed is:

1. A computer system comprising:

at least one input device;

an output device;

a memory device;

a processing unit coupled to the input device, the output device and the memory device, the processing unit executing a computer program stored in the memory device, the computer program controlling the output device to display a graphical user interface and controlling the at least one input device to accept input to manipulate a view of objects displayed within the graphical user interface;

the graphical user interface displaying objects derived from hierarchically arranged data, wherein each object displayed is graphically scaled in size and proportion and positioned in relation to other objects within a hierarchy of the data.

2. The computer system of claim 1, wherein each object displayed on the graphical user interface is depicted graphically as being interconnected to other objects to which it is directly related to in a hierarchy of the data, so as to form a graphical representation of the hierarchical nature of each object as it relates to other objects.

3. The computer system of claim 1, wherein an object hierarchically related directly above another object in the hierarchy is a parent object and the graphical user interface includes at least one centrally located parent and an object hierarchically related directly below an object in the hierarchy is a child object and is displayed radially adjacent to the parent object; and wherein the input device may be used to select a child object on the graphical user interface and, in response, the computer program will generate a zoom-in view of that child object including any child objects related to that child object.

4. The computer system of claim 3 wherein

the graphical user interface displays a first view of individually selectable hierarchically arranged objects including a parent object and a plurality of child objects peripherally arranged about the parent object; and

upon a zoom-in selection of a child object with the at least one input device, the processing unit executes the computer program to transform the display of the graphical user interface to a zoomed-in view in which the parent object is relocated and proportionally increased in size to appear as a grand-parent object and the child object selected is relocated and proportionally increased in size to appear as a centrally located parent object and wherein previously undisplayed child objects that are hierarchically related below the child object selected become displayed and peripherally arranged about the centrally located parent object.

5. The computer system of claim 4, wherein upon selection of the child object, the grand-parent object becomes no longer displayed and the centrally located parent object is relocated and proportionally increased in size to appear as a new-grand-parent object.

6. The computer system of claim 1, wherein an object hierarchically related directly above another object is a parent object and an object hierarchically related directly below another object is a child object; and wherein the input device may be used to select a parent object and, in response,

the computer program will generate a zoom-out view of that parent object by removing any child objects previously displayed and by displaying all child objects related to the selected parent object to a specific level of detail in the hierarchy of data.

7. The computer system of claim 6 wherein

the graphical user interface displays a first view of individually selectable hierarchically arranged objects including a grand-parent object, a centrally located first parent object and a plurality of child objects peripherally arranged about the centrally located parent object; and

upon a zoom-out selection of the grand parent object with the at least one input device, the processing unit executes the computer program to transform the display of the graphical user interface to a zoomed-out view in which child objects disappear from the graphical user interface and the first parent object is proportionally reduced in size to appear as a child object and the grand-parent object is relocated and proportionally reduced in size to appear as a centrally located parent object and the child object and other previously undisplayed child objects that are hierarchically related below the centrally located parent object become displayed and peripherally arranged about the centrally located parent object.

8. The computer system of claim 7, wherein upon selection of the grand parent object, an object hierarchically above the grand parent object becomes displayed as a new grand parent object.

9. The computer system of claim 1, wherein the processing unit executes the computer program to allow the input device to specify a number of hierarchy levels to be displayed and the computer program controls the output device to display, on the graphical user interface, a total number of objects that is dependent upon the number of hierarchy levels specified.

10. The computer system of claim 9, wherein the graphical user interface further includes a locale map providing a user with an indication of where in the hierarchy the objects exist that are currently on display in the graphical user interface.

11. The computer system of claim 1, wherein the hierarchically arranged data associated with any object displayed represents data associated with a managed device on a computer network.

12. The computer system of claim 11, wherein the color of an object on the graphical user interface depicts information concerning the state of the managed device which the object represents.

13. The computer system of claim 12, wherein if an object having associated corresponding data in the hierarchy contains a fault, the fault is reflected in the data for that object in the hierarchy of data and is reflected to objects above that object in the hierarchy of data such that, in the graphical user interface, objects that are displayed and that are hierarchically related above the faulted device include a visible indication of the presence of the fault located below that object, thus providing an indication on the graphical user interface of a selected object path to follow to find the fault.

14. The computer system of claim 1, wherein the objects displayed appear circular in nature and wherein the computer program employs fractal geometry to properly display each object without overlapping other objects and properly

scales each object in relation to a hierarchy of all objects in the hierarchically arranged data.

15. A graphical user interface representing objects having a hierarchical relationship to one another, the interface providing:

a top level view of an entire hierarchical network including a centrally located parent object associated with an upper hierarchical level, child objects associated with a level below that of the parent object, the child objects peripherally arranged radially adjacent to the parent object and scaled proportionally to the parent object to reflect the hierarchical relationship between the parent and child objects;

wherein each object is selectable to present a zoomed-in view of the hierarchical relationship of object at a hierarchical level of the object selected with other child objects hierarchically related below the selected object; and

wherein each object is positioned relative to the other objects such that no two objects graphically occlude one another.

16. The graphical user interface of claim 15 wherein, upon selection of an object, providing an animated representation of transition from one zoom view to another, such that hierarchy information concerning the relationship of a selected object to objects above and below the selected object in the hierarchy is conveyed to the user by resizing the selected object to be a centrally located object and by resizing objects hierarchically related above and below the selected object in relation to their hierarchical positions so as to surround the selected centrally located object.

17. A method for representing and navigating among objects of a hierarchical network comprising the steps of:

providing a user with a display of a plurality of objects having a hierarchical relationship with one another, including at least one parent object, at least one child object hierarchically related below the at least one parent object, each child object placed equi-distant from the parent object and radially adjacent thereto, and grand-child objects hierarchically related below the at least one child object and positioned equidistant and radially adjacent thereto;

allowing a user to select an object, and depending upon the objects position within the hierarchy, zooming in on that object as a centrally located object and displaying objects directly related above and below that object on the display.

18. A method of displaying hierarchical data comprising the steps of:

determining a node to be displayed within a hierarchy of the hierarchical data;

creating a graphical object representing the node;

rendering the graphical object;

determining child nodes which depend from the node, and for each child node, creating a child graphical object representing the child node and computing child node characteristics for the child graphical object and rendering the child graphical object;

rendering links between the child node and the node;

determining each child node created, and passing each child node as a new top node to the step of determining

a node to be displayed and repeating the processing steps of creating and rendering for each child node as a top node until a number of levels of hierarchical data are displayed.

19. A method for displaying hierarchical data comprising the steps of:

i: displaying a parent node;

ii: displaying a plurality of child nodes peripherally arranged radially adjacent to the parent node, each child node depending from the parent node in the hierarchy and each child node being scaled in size proportionally to a level in the hierarchical data to which the child node corresponds;

iii: displaying a plurality of data links from each of the plurality of child nodes, each data link extending to the parent node from which that child node depends in the hierarchical data.

20. The method of claim 19, further including the step of allowing a user to navigate through the hierarchical data by selecting at least one child node and repeating steps i, ii and iii with the parent node set equal to the child node selected by the user.

21. A computer program product having a computer readable medium including computer program logic recorded thereon that, when executed on a computer processor, performs the steps of:

providing a user with a display of a plurality of objects having a hierarchical relationship with one another, including at least one parent object, at least one child object hierarchically related below the at least one parent object, each child object placed equi-distant from the parent object and radially adjacent thereto, and grand-child objects hierarchically related below the at least one child object and positioned equidistant and radially adjacent thereto;

allowing a user to select an object, and depending upon the objects position within the hierarchy, zooming in on that object as a centrally located object and displaying objects directly related above and below that object on the display.

22. A computer program product having a computer readable medium including computer program logic recorded thereon that, when executed on a computer processor, performs the steps of:

determining a node to be displayed within a hierarchy of the hierarchical data;

creating a graphical object representing the node;

rendering the graphical object;

determining child nodes which depend from the node, and for each child node, creating a child graphical object representing the child node and computing child node characteristics for the child graphical object;

rendering links between the child node and the node;

determining each child node created, and passing each child node as a new top node to the step of determining a node to be displayed and repeating processing of the creating and rendering each child node as a top node until a number of levels of hierarchical data are displayed.

* * * * *